Europäisches Patentamt | European Patent Office | Office européen des brevets

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.:
Application no.: 02425440.1
Demande n°:

Anmeldetag:
Date of filing: 03/07/02
Date de dépôt:

Anmelder:
Applicant(s):
Demandeur(s):
STMicroelectronics S.r.l.
20041 Agrate Brianza (Milano)
ITALY

Bezeichnung der Erfindung:
Title of the invention:
Titre de l'invention:
Code compression process, system and computer program product therefor

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: State: Pays:	Tag: Date: Date:	Aktenzeichen: File no. Numéro de dépôt:

Internationale Patentklassifikation:
International Patent classification:
Classification internationale des brevets:
H03M7/30, G06F9/45, H03M7/40

Am Anmeldetag benannte Vertragstaaten:
Contracting states designated at date of filing: AT/BG/BE/CH/CY/CZ/DE/DK/EE/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/
Etats contractants désignés lors du depôt:

Bemerkungen:
Remarks:
Remarques:
For the original title, see page 1 of the description

"Procedimento per la compressione di codici, relativo sistema e prodotto informatico"

* * *

Campo dell'invenzione

La presente invenzione si riferisce alle tecniche per la compressione di codici.

L'invenzione è stata sviluppata con particolare attenzione alla possibile applicazione ai sistemi di tipo embedded. Il riferimento a tale possibile applicazione non deve essere in alcun modo interpretato come limitativo della portata dell'invenzione, che è affatto generale.

Descrizione della tecnica nota

Nell'implementazione dei processori, ed in particolare dei processori per sistemi embedded, è necessario tenere in conto diverse limitazioni: fra queste, le limitazioni principali sono date dai costi, dall'assorbimento di potenza e dall'area di silicio occupata.

Le limitazioni in termini di costo sono dettate dalla volontà di utilizzare tali processori come dispositivi di controllo in sistemi digitali di tipo consumer (ad esempio per applicazioni tipo smart card) dove ad elevati volumi di produzione si devono accompagnare bassi costi per unità prodotta.

L'aspetto dell'assorbimento di potenza sta diventando sempre più importante, anche in considerazione dell'applicazione di tali processori a dispositivi quali telefonici cellulari, cosiddetti portable digital assistant o PDA, ecc., dove è importante prolungare quanto più possibile la durata della carica delle batterie.

Per quanto riguarda l'area di silicio occupata - o in generale la dimensione del chip (die size) - se si pensa in particolare ad applicazioni del tipo smart

card risulta evidente che è imperativo dare origine a piccole aree di occupazione.

Dall'altra parte, le applicazioni oggi correnti richiedono un livello sempre più elevato di flessibilità per essere in grado di soddisfare a tutte le esigenze funzionali richieste. Questo fatto fa sì che sia necessario caricare nelle memorie del sistema, per eseguirli sul relativo processore (ad esempio DSP o CPU) porzioni di codice software sempre più rilevanti; fatto, questo, che si traduce nella tendenza ad utilizzare memorie costose e suscettibili di assorbire livelli rilevanti di potenza.

Sussiste quindi l'esigenza di fornire soluzioni per ridurre le dimensioni della memoria di programma di un dispositivo del tipo sopra descritto.

Una soluzione di questo tipo è quella nota con il nome commerciale di IBM CodePack(TM). Al riguardo si può utilmente consultare il relativo manuale d'uso denominato IBM CodePack(TM) Power PC Code Compression Utility, User's manual version 4.1.

L'algoritmo in questione è stato messo a punto con il fine principale di migliorare le prestazioni del nucleo o core PowerPC. L'architettura del core in questione, progettata come una tipica architettura RISC (Reduced Instruction Set Computer), incorpora istruzioni di lunghezza fissa per semplificare il progetto di processori con prestazioni elevate. Quasi sempre, le istruzioni di lunghezza fissa forniscono una densità di codice inferiore rispetto ad un progetto di tipo CISC (Complete Instruction Set Computer), ma per la maggior parte delle applicazioni di tipo embedded, per cui il costo è un fattore molto significativo, la densità di codice è un parametro di progetto importante.

Lo schema di compressione adottato nella soluzione IBM in questione è basato sull'osservazione del fatto che le istruzioni PowerPC utilizzate nelle applicazioni reali non sono distribuite in modo uniforme su tutti i 32 bit utilizzati per rappresentata una singola istruzione. Talune configurazioni o pattern di istruzioni appaiono con frequenza molto maggiore rispetto ad altre. Dal punto di vista concettuale, queste istruzioni possono essere rappresentate tramite speciali sequenze di bit, più corte rispetto alle istruzioni originali, e sostituite con le istruzioni originali dal controllore di decompressione quando le istruzioni vengono portate nel core del processore.

E' possibile pensare a diverse varianti di questo schema di base. Un perfezionamento che si dimostra efficace nel caso delle istruzioni PowerPC su 32 bit è quello di comprimere separatamente i 16 bit in posizione "alta" ed i 16 bit in posizione "bassa" dei 32 bit che compongono l'istruzione.

Per aumentare la densità del codice, le istruzioni utilizzate in modo più comune devono essere sostituite da sequenze di bit più piccole, destinate ad essere espanse successivamente dal controllore di decompressione così da ricostituire le istruzioni originarie PowerPC. Per massimizzare la densità di istruzioni si impiegano sequenze con una lunghezza variabile, in termini di numero di bit, collocate in memoria in condizioni "end-to-end". Il software di compressione tratta l'immagine del codice non-compresso un'istruzione per volta, esaminando l'istruzione così da identificare corrispondenze (match) nell'ambito del contenuto della tavola di decodifica.

Come già si è detto, questa tecnica nota viene applicata separatamente ai 16 bit in posizione "alta" ed ai sedici bit "bassa" nell'ambito dell'istruzione, e

gli n valori alti più frequenti sono in generale diversi rispetto agli n valori "bassi" più frequenti.

Nell'ambito della soluzione nota qui considerata si riscontra però che, tanto con riferimento ai bit "alti" quanto con riferimento ai bit "bassi", se si riscontra una stringa di 16 bit tale da non ricorrere in modo sufficientemente frequente da ricevere una specifica posizione (slot) nell'ambito della tabella, la stringa di bit in questione deve essere riprodotta in formato "letterale" su 16 bit. Ciò richiede l'aggiunta di tre bit (di solito "111") per identificare appunto tale stringa come una stringa di bit che non può essere compressa.

Si determina quindi una situazione - abbastanza incongruente - per cui l'applicazione di un meccanismo destinato ad attuare una compressione fa sì che per rappresentare i 16 bit originali se ne debbano usare ben 19. La situazione è resa ancora più incongruente quando le condizioni di non-comprimibilità considerate in precedenza si manifestano tanto per i 16 bit "alti" quanto per i 16 bit "bassi" che compongono un'istruzione. Questa istruzione, originariamente comprendente 32 bit finisce infatti per essere rappresentata con 38 bit. Di conseguenza l'istruzione "compressa" non risulta affatto tale in quanto comprende ben sei bit in più rispetto all'istruzione originaria.

Il formato di memorizzazione dell'istruzione compressa prevede, nell'ordine, i tag (essenzialmente i campi che identificano il tipo di istruzione e le modalità utilizzate per la compressione rispettivamente per i bit "alti" e per i bit "bassi") e poi i valori di indice - ossia, in pratica, la rappresentazione in forma compressa (anche qui, nell'ordine, per i bit "alti" e per i bit "bassi").

Se tutte le istruzioni compresse sono destinate ad essere seguite in sequenza, questo schema di codifica relativamente semplice risulta soddisfacente. Il controllore di decompressione viene infatti semplicemente puntato all'inizio dell'immagine dell'istruzione compressa per poi andare a recuperare (fetch) e decomprimere il flusso dell'istruzione così come richiesto dal processore.

Tuttavia, esistono flussi di istruzioni di applicazione che contengono un numero significativo di ramificazioni.

Questi flussi richiedono al controllore di decompressione di localizzare l'inizio delle istruzioni compresse in qualunque luogo della regione di indirizzamento rappresentata dall'immagine di istruzione compressione rendono più critico il funzionamento di questo schema.

Infatti, per riuscire a localizzare l'inizio di una qualunque istruzione, è necessario disporre di un meccanismo in grado di tradurre l'indirizzo dell'istruzione desiderata in un indirizzo che punta verso l'inizio della corrispondente istruzione complessa.

Lo schema utilizzato nella soluzione nota cui si sta qui facendo riferimento prevede di suddividere le istruzioni in gruppi e di fornire un meccanismo in grado di localizzare l'inizio del gruppo che contiene l'istruzione desiderata.

Il gruppo è successivamente trattato dal meccanismo di decodifica fino a quando si raggiunge l'istruzione desiderata. Questo meccanismo richiede però un meccanismo di indirizzamento implementato tramite una tabella di indici, chiamata di solito address translation table (ATT), ovverosia tabella di traduzione degli indirizzi.

Scopi e sintesi della presente invenzione

La presente invenzione si prefigge lo scopo di fornire una soluzione di compressione in grado di superare gli inconvenienti delineati in precedenza.

Secondo la presente invenzione, tale scopo viene raggiunto grazie ad un procedimento avente le caratteristiche richiamate in modo specifico nelle rivendicazioni che seguono. L'invenzione riguarda anche il relativo sistema nonché il corrispondente prodotto informatico direttamente caricabile nella memoria di un processore digitale e contenente porzioni di codice software suscettibili di attuare il procedimento secondo l'invenzione quando il prodotto viene eseguito su un processore digitale.

La soluzione secondo l'invenzione offre una tecnica di compressione suscettibile di operare direttamente sul codice binario generato dal compilatore per comprimerlo e generare un nuovo file binario le cui dimensioni risultano circa del 40% inferiori rispetto alle dimensioni del file originario.

Tale file compresso è quindi suscettibile di essere caricato nella memoria di programma del processore.

Il decodificatore è suscettibile di essere realizzato di preferenza sotto forma di un piccolo ASIC suscettibile di essere collocato fra il bus di sistema e la memoria cache di istruzioni (ad esempio nell'unità di controllo di memoria o MCU o, se è presente una gerarchia di cache con almeno due livelli di cache, fra il primo ed il secondo livello, così da mascherare meglio la sua latenza), ovvero ancora direttamente al terminale di uscita della memoria esterna.

Breve descrizione dei disegni annessi

L'invenzione sarà ora descritta, a puro titolo di esempio non limitativo, con riferimento ai disegni annessi, nei quali:

- le Figure 1 e 2 illustrano la costruzione dei vocabolari e la gerarchia dei dati utilizzabili in una soluzione secondo l'invenzione,
- le Figure 3 a 7 comprendono vari diagrammi utili per comprendere il meccanismo posto alla base della soluzione secondo l'invenzione,
- le Figure 8 e 9 illustrano vari schemi di organizzazione dei flussi di dati in una soluzione secondo l'invenzione,
- la Figura 10 è una prima rappresentazione del processo di compressione dei dati in una soluzione secondo l'invenzione,
- le Figure 11 a 13 rappresentano, in modo complementare, il processo di decodifica,
- le Figure 14 a 16 rappresentano varie possibili implementazioni del meccanismo di decodifica delle parole codificate, e
- le Figure 17 e 18 rappresentano, sotto forma di schema a blocchi funzionali, due architetture di elaborazione suscettibili di utilizzare la soluzione secondo l'invenzione.

Descrizione particolareggiata di esempi di attuazione della soluzione secondo l'invenzione

Con riferimento allo schema della figura 1, si considera di partire da una sequenza non-compressa di N istruzioni (da 0 a N-1) che formano un programma originario OP destinato ad essere eseguito da un processore, così come scritto dal compilatore. Ciascuna istruzione si suppone essere lunga n bit (dove n è un multiplo di due).

E' allora possibile pensare di suddividere il flusso o stream di dati in ingresso in d flussi diversi di sotto-istruzioni ottenuti dal flusso originale prendendo da esso d parti diverse. La somma di d sotto-istruzioni omologhe (che appartengono alla stessa istruzione originaria non compressa su n bit) è pari a n (con $n_1+n_2+...+n_k+...n_m=n$), dove n_k rappresenta il numero di bit compresi nella k-esima sotto-istruzione.

Nel seguito si considererà per semplicità il caso in cui n è pari a 32 bit, m è pari a 2 e $n_1=n_2$ è uguale a 16 bit, cosicché nell'ambito della singola istruzione sono distinguibili una porzione "alta" (high) ed una porzione "bassa" (low) e che le due parti "alta" e "bassa" comprendano ciascuna lo stesso numero di bit, ad esempio 16 bit nel caso in cui il numero n di bit compreso in ciascuna istruzione sia pari a 32.

Si apprezzerà che la scelta di tali specifici valori, ed in particolare la partizione della singola istruzione in due porzioni identiche fra loro, non è in alcun modo imperativa ai fini dell'attuazione dell'invenzione dal momento che quanto detto nel seguito con riferimento a questo caso specifico (citato in via principale per semplicità di illustrazione) si applica ad una partizione qualsiasi.

Sulla base di considerazioni statistiche è possibile rilevare che la densità di probabilità delle occorrenze di ciascuna semi-istruzione segue una curva laplaciana. Per questo motivo, è possibile calcolare due diversi vocabolari che contengono le M istruzioni più frequenti per ciascuna metà del flusso originario di istruzioni.

Naturalmente, queste M istruzioni (dove M è molto più piccolo di N) non sono tali da coprire l'insieme totale delle istruzioni di programma. Di conseguenza è necessario riservare una delle M parole disponibili in

ciascun dizionario o vocabolario come via di fuga o "escape" ed associare a questa particolare parola una probabilità di occorrenza pari alla somma delle occorrenze di tutte le parole che non appartengono agli insiemi del vocabolario e poi riordinare nuovamente tutte le parole, trattando la parola denominata "escape" come le altre.

Si può notare che un buon candidato per M è un multiplo di 2 (ad esempio si può prendere $M=2^{10}-1=1023$, dove M-1=1022 sono le parole reali, ed una è la parola che funge da escape).

Nello schema della figura 1, il riferimento CD rappresenta i vocabolari completi formati secondo i criteri descritti in precedenza e comprendenti rispettivamente H e V parole per le porzioni "alta" e "bassa". Infine, il riferimento FD indica i vocabolari finali, comprendenti M parole, anch'essi suddivisi in una parte "alta" ed in una parte "bassa", con la rappresentazione della parola di escape in entrambi i casi.

Una volta identificati i vocabolari, come descritto in precedenza, risulta possibile sostituire ciascuna semi-istruzione del codice con il corrispondente indice da 0 a M-2, se presente, o con il codice di escape seguito dalla semi-parola originaria. Naturalmente la larghezza dell'indice, espressa in bit ed indicata con b, deve soddisfare la seguente equazione:

$$b = \log_2(M+1)$$

Nell'esempio indicato in precedenza, con M pari a 1023, b è pari a 10 bit.

A questo punto, così come illustrato nella figura 5, la sequenza degli indici è raggruppata in un gruppo di vettori GV aventi dimensioni (espresse in termini di

istruzioni per vettore) pari alla dimensione della line cache del processore.

Ciascun gruppo di vettori GV è a sua volta suddiviso in B vettori che rappresentano il livello più basso della gerarchia e contenenti ciascuna un numero di indici pari a nI, dove nI è dato dal rapporto CACHE_LINE/B fra la lunghezza della cache di linea del processore e B, dove B e nI sono potenze di due. Ciascun vettore è codificato in modo indipendente dagli altri.

Alternativamente, per ciascun gruppo di vettori, alla fine del processo di codifica si calcola e si salva in una tabella, chiamata tabella di traduzione degli indirizzi (ATT), l'indirizzo di partenza su 32 bit del blocco compresso, oppure le differenze, espresse in byte, rispetto all'ultimo indirizzo completo riportatovi. Tale tabella è poi caricata nella memoria interna del decodificatore. Questa tabella permette una gestione efficiente dei salti presenti nel codice, consentendo una rapida conversione degli indirizzi.

Per codificare ciascun vettore, è necessario conoscere la norma degli indici contenuti all'interno. Per norma si intende la grandezza definita come

$$norma = \sum_{nI}(index)$$

Le dimensioni, in bit, della norma dipendono dalle dimensioni dei vocabolari e in particolare è pari a nI+b bit.

Continuando con l'esempio fatto in precedenza, in cui b è pari a 10 bit (M=1023) e scegliendo la dimensioni del vettore pari a quattro indici, la dimensione massima, espressa in bit, della norma è pari a 12 bit (2^{12}=4096 > 1023*4=4092).

Poiché questa componente del vettore codificato può essere molto importante, un modo per ridurre la

quantità di bit necessari può essere quella di utilizzare una codifica di lunghezza variabile (variable length coding o VLC) solo per la norma. Le tabelle di look-up della codifica VLC, diverse rispettivamente per la parte alta e la parte bassa del codice, possono essere determinate tramite diverse simulazioni su campioni (benchmark) diversi e rilevando la ricorrenza di ciascuna norma possibile al variare dei parametri dati dalla dimensione del vocabolario e da B.

A partire da questi dati è possibile determinare la curva di distribuzione per ciascuna combinazione dei due parametri. Queste curve sono divisibili in nP parti con la stessa probabilità (vale a dire con la stessa area) con numeri elementi all'interno di ciascuna area arrotondati a multiplo di due più vicino.

A titolo di esempio, la figura 3 fa vedere un caso per M = 1023 e nI = 4. La curva rappresenta la distribuzione statistica della norma per la parte alta del codice su un ampio insieme di campioni di prova. Le due linee verticali esprimono la divisione della curva in tre parti con la stessa probabilità in corrispondenza di valori di norma pari rispettivamente a 64 e 320.

In questo esempio, i valori di norma compresi fra 0 e 64 sono codificati con un primo bit di etichetta (0, ad esempio) e sei bit che rappresentano un valore reale. I successivi 256 valori sono rappresentati con due bit di etichetta (10, per esempio) e otto bit per il valore, ed infine gli altri 3772 valori (possibili, ma poco probabili) con due bit di etichetta (11, per esempio) e dodici bit per i valori.

Facendo ricorso alla distribuzione statistica laplaciana degli indici dei vocabolari è possibile definire come L il numero di indici successivi

codificati insieme come parola di codice unica (sempre una potenza di due, fra 1 e 32) ed una tabella di look-up, chiamata nel seguito anche N_table, con dimensioni pari a [L, K], dove L è pari alle parole che verranno concatenate insieme e K è la massima norma possibile (somma dei valori assoluti) per le parole codificate destinata ad essere pre-calcolata così come meglio descritto nel seguito, senza che ci sia però una dipendenza dai dati di ingresso.

La parola codificata ottenuta partendo dagli L successivi indici su 16 bit, qui denominata *encoded*, è calcolata nel modo seguente:

$$encoded = \sum_{i=1}^{L} \sum_{j=1}^{x} Ntable[i,j]$$

dove la prima sommatoria si estende per i che va da 1 a L e la seconda sommatoria si estende per j che va da 1 a x_i.

Di conseguenza, il numero di somme necessarie per codificare L indici è pari a $L*\max(x_j)$.

Se si ritorna all'esempio visto in precedenza, e supponendo che L sia pari a 4, il numero massimo di somme necessarie per ottenere una parola codificata è pari a 4*1023=4092.

Inoltre, per un valore di norma dato è possibile predeterminare il numero di bit necessari per la parola codificata contando il numero di bit necessari utilizzando la seguente equazione:

bit = *count*(*Ntable*[L+1.*norma*])

dove "*count*" è una funzione che conta il numero di bit richiesti dal suo parametro di ingresso. Questa funzione è implementabile ad esempio con il seguente pseudocodice:

```
ui32_t count(ui32_t input_number)
{
```

quantizzatore vettoriale piramidale introdotto da Thomas Fischer nell'articolo "A Pyramid Vector Quantizer" apparso su "IEEE Transactions on Information Theory", vol. IT-32, 4 luglio 1986.

Nell'articolo in questione è descritto ed illustrato un procedimento per sfruttare le proprietà di una sequenza di variabili indipendenti con una distribuzione laplaciana.

Definendo $X=\{x_1, x_2, \ldots, x_L\}$ un vettore di variabili casuali x_i (gli indici di vocabolario, in questo caso) la densità di probabilità:

$$f_x(a) = \frac{\lambda}{2} e^{-\lambda y}$$

con valore medio nullo e varianza

$$\sigma^2 = 2/\lambda^2$$

la densità di probabilità delle x_i risulta nell'insieme

$$f_x(a) = \prod_{i=1}^{L} f_x(a_i) = (\frac{\lambda}{2})^L e^{-\lambda y}$$

dove la moltiplicatoria si estende per i che va da 1 a L e r rappresenta la norma del vettore X

$$r = \sum^{L} |x_i| = ||X||_1$$

Il parametro r specifica un contorno con densità di probabilità costante. Per una sorgente laplaciana la media risulta pari a

$$E[r] = \frac{L}{\lambda}$$

per cui è possibile definire una piramide nel modo seguente

$$S(L,Q) = \{\forall X \in \Re^L: \sum_{i=1}^{L} |x_i| = ||X||_1 = Q\}$$

La variabile casuale r può essere considerata come il parametro che rappresenta la superficie di una particolare piramide S(L,r).

```
        ui32_t bits;
        for(bits=1;;bits+++)
        {
            if((input>>=1)==0)
        }
        return bits;
    }
```

Quando uno o più codici di escape sono stati codificati in una singola parola di codice, nello stream di uscita dopo di essa è collocata la parola originaria su 16 bit non compressa.

Di conseguenza, il formato dell'istruzione compressa risulta essere quella illustrato nella figura 4 dove compaiono nell'ordine:

- l'etichetta (tag) di norma "alta" (HNT),
- l'etichetta di norma "bassa" (LNT),
- l'indice di norma "alta" (HNI),
- l'indice di norma "bassa" (LMI),
- gli indici L "alti" (HLI),
- le parole di escape "alte" (HESCW),
- gli indici L "bassi" (LLI), e
- le parole di escape "basse" (LESCW).

Ritornando a parlare della tabella N_table considerata in precedenza, si è già notato che la stessa ha una matrice di dimensioni [L, K], dove L è il numero di parole concatenate in una parola di codice e K è il valore massimo disponibile per la norma, dipendente tanto dalle dimensioni dei vocabolari quanto dalla parola concatenata.

Sempre facendo riferimento ad un esempio, se L è pari a 4 ed i vocabolari sono composti da 1024 parole, la dimensione di N_table è pari a [1023*4].

In questo contesto di distribuzione statistica laplaciana è utile far riferimento al concetto di

Naturalmente, qualunque tipo di soluzione utilizzata per comprimere la memoria di programma di qualunque tipo di processore o DSP deve essere di tipo senza perdite o lossless. Deve quindi essere possibile recuperare tutte le istruzioni originarie, senza perdite di informazione: in caso contrario, il relativo programma non può operare in modo corretto.

Questa limitazione piuttosto severa, suscettibile di essere allentata nel caso della compressione di dati, implica che in linea di principio la soluzione può generare più bit rispetto a quelli necessari nella parola "non compressa".

Naturalmente questo non è accettabile in termini generali, ciò vale ancora di più se si pensa a come il processore recupera le istruzioni dalla memoria di programma. Normalmente, nella funzione di fetch il processore ricupera un'intera linea di cache di istruzioni in un colpo solo: come esempio si immagini una cache di riga lunga 16 istruzioni a 32 bit, il che significa 64 byte.

Il caso può essere naturalmente esteso a qualunque dimensione di linea di cache.

La soluzione implementata tiene conto di questo fatto, cercando di comprimere un gruppo di vettori di istruzioni in una fase di pre-analisi e andando a verificare quanti bit sono necessari. Se i bit generati sono più di quelli nella condizione non-compressa, è previsto che vengano copiate semplicemente le 16 istruzioni, così come sono. Solo in caso contrario viene utilizzato il procedimento di compressione.

Per una migliore comprensione di quanto detto nel seguito può essere utile far dapprima riferimento alla fase di decompressione, tenendo conto che una delle principali caratteristiche richieste per la decompressione del codice è quella di essere in grado

Definendo quindi N_table (L,K) come il numero di punti interi appartenenti alla superficie S(L.K) si vede che, ad esempio, nella situazione rappresentata nella figura 5, i punti interi sulla superficie della piramide S(3,4) sono illustrati con riferimento ad una piramide con L=3 dimensioni con norma K=4. A questa piramide appartengono soltanto i punti interi la cui somma di coordinate (in valore assoluto) è pari a 4.

La costruzione della tabella deve soddisfare all'equazione

$$N(i,j) = \sum_{i=1}^{L} \sum_{j=1}^{K} N(i-1,j)+N(i,j-1)$$

dove la prima sommatoria si estende per i che va da 1 a L e la seconda sommatoria si estende per j che va a K. Per L pari a 4 e dimensioni di vocabolario è pari a 1023 si ottiene la superficie rappresentata nella figura 6.

In questa situazione si permette a ciascun valore x_i (l'indice di vocabolario nella parola concatenata da codificare) di essere compreso fra 0 e L volte la dimensione del vocabolario. Questo non è però un dato realistico per cui è più corretto prevedere che ciascun x_i possa variare soltanto fra 0 e la dimensione del vocabolario (definita nel seguito D).

L'equazione vista in precedenza si modifica in modo conseguente nel modo qui sotto riportato

$$N(i,j) = \sum_{i=1}^{L} \sum_{j=1}^{K} \sum_{q=0}^{\max(D,j)} N(i,j)+N(i-1,j-q)$$

dove le tre sommatorie si estendono, nell'ordine, per i che va da 1 a L, j che va da 1 a K e q che va da 0 a max(D,j).

In questo caso, per L pari a 4 e dimensioni di vocabolario pari a 1023 si ottiene la superficie di tabella di norma ridotta rappresentata nella figura 7.

codificata che rappresenta le n istruzioni formanti un vettore compresso.

Passando, con specifico riferimento alle figure 8 e 9 all'illustrazione dettagliata dell'organizzazione del flusso si può - sempre a titolo di esempio - supporre di operare in una condizione in cui:

- la "profondità" dei vocabolari è pari a 2 vocabolari, ciascuno con profondità di 16 bit,
- la dimensione massima del vocabolario è pari a 1023 parole, compreso il codice di escape,
- il numero di parole concatenate che formano un vettore è pari a 4.

In queste condizioni, la norma massima teorica che sarebbe possibile avere per una parola codificata (un semivettore, così come già evidenziato in precedenza) è 1023*4=4092. Lo scopo è quello di ridurlo ad un valore inferiore rispetto a 31.

Invece di codificare tale norma tramite una codifica VLC, come precedentemente ipotizzato, questo obiettivo può essere conseguito facendo avanzare nel flusso di bit il bit meno significativo di ciascuna parola che compone la parola di codifica e effettuando poi una divisione per un fattore 2 (il che equivale ad uno shift verso destra dei bit), continuando poi in modo ricorsivo fino a quando la norma è inferiore a 32 (in questo caso essa può essere sempre codificata con 5 bit).

Ad esempio, supponendo di avere una parola da codificare con i seguenti indici {15, 34, 18, 43} con norma W=15+34+18+43=110 è possibile procedere nel modo seguente

{15, 34, 18, 43}, W=110
{7, 17, 9, 21}; forw={1, 0, 0, 1}; W=(110-2)/2=54
{3, 8, 4, 10}; forw={1, 1, 1, 1}; W=(54-4)/2=25

di accedere a qualunque istruzione del codice, seguente il flusso di istruzioni richieste dal programma che viene eseguito. Questo flusso è, in generale, diverso rispetto alla sequenza statica delle istruzioni generate dal compilatore. Questo per la presenza di salti, loop e chiamate di funzioni.

L'adozione della tabella di traduzione degli indirizzi ATT citata in precedenza consente di mappare l'indirizzo di un'istruzione non compressa nel codice compresso così da poter decomprimere qualunque istruzione del programma senza dover di necessità decomprimere tutte le istruzioni precedenti.

In sostanza, la soluzione secondo l'invenzione si basa su una gerarchia dei dati finali riassumibile nei termini seguenti:

Gruppo di compressione	2 macroblocchi consecutivi (compresso)
Gruppo di vettori	R blocchi consecutivi R = dimensione riga di cache / L (compresso)
Vettore	L istruzioni consecutive su 16 bit con L pari al divisore intero della dimensione della riga di cache (compresso)
Istruzione	singola istruzione su 16 bit (non compresso)

Sulla base del meccanismo descritto in precedenza si è così creata una particolare struttura del flusso compresso dove, sostanzialmente, si è perseguito lo scopo di ridurre la dimensione della norma della parola

bit ed il processo di decodifica della parola codificata.

In questa soluzione alternativa, corrispondente ad un'organizzazione ottimale dello stream, così come rappresentata nella figura 9, è prevista, nell'ordine, la collocazione delle stringhe di bit, corrispondenti, rispettivamente, alle grandezza SS, N, EW e SB viste in precedenza.

Questa composizione dello stream, che non influenza il fattore di compressione, semplifica notevolmente il processo di decompressione.

Nel caso migliore, si utilizzano infatti solo 8 bit per ciascun semivettore, tenuto tale di 8 byte per gruppo di vettori (16 istruzioni). Al contrario nel caso peggiore (peraltro molto raro) si utilizzano (3+7*4+5+13)*2*4=140 bit ossia 17,5 byte.

Se necessario, si aggiungono le parole di escape su 16 bit. In questo caso si aggiungono 16*16*2 bit ossia 64 byte per un totale 64+17,5=81,5 byte.

In ogni caso, questo caso può essere evitato dalla pre-analisi, per cui soltanto i 64 byte originali vengono codificati.

Si noti ancora che non è necessario che il codice di escape venga associato all'indice più grande nel vocabolario. In generale questo può essere vero per la parte "alta" su 16 bit nelle istruzioni non compresse, poiché questa parte contiene l'operando e il codice operativo e la loro variabilità non è molto alta.

Al contrario, con la parte di 16 bit più bassa delle istruzioni la situazione può essere molto diversa (fatto dimostrato nella pratica), poiché ci sono gli operandi delle istruzioni e così la loro variabilità risulta più elevata facendo sì che (in funzione della ricorrenza) l'indice associato possa essere più basso.

così da raggiungere il risultato con due shift verso destra (S).

Nella formula sopra riportata con forw si sono indicati gli insiemi di bit che rappresntano i resti da aggiungere in fase di decodifica e che vanno inseriti (forwarded) all'interno dello stream compresso.

Si può notare che nel caso peggiore ({1023, 1023, 1023, 1023}; W=4092) il numero complessivo di shift è pari a 7, con una norma residua pari a 28 e la parola da codificare {7, 7, 7, 7}, per cui il numero di shift da realizzare può essere sempre codificato con 3 bit. La figura 8 rappresenta lo stream di bit per un semivettore, così come rappresentato comprende quindi:

- 3 bit indicativi degli shift effettuati (SS),
- il numero di bit shiftati (SB) pari, nell'esempio qui illustrato, a 4 bit per 2 shift complessivi,
- la norma (N), rappresentata su 5 bit, e
- la parola codificata (EW).

Peraltro, quello illustrato nella figura 8 può non essere il modo migliore per raccogliere i numeri nello stream compresso pensando alla funzione del decodificatore.

Invece di prevedere ad esempio S campi di 4 bit, che indicano i resti, shift dopo shift, è possibile mettere direttamente il totale dei 4 resti, di S bit ciascuno, considerando il fatto che si divide sempre per 2.

Così, con riferimento all'esempio visto in precedenza, è possibile porre {1, 0, 0, 1}*2+{1, 1, 1, 1}={3, 1, 1, 3}.

Inoltre, i S*4 bit relativi ai resti delle divisioni possono essere vantaggiosamente collocati alla fine dello stream. Questo consente al decodificatore una migliore lettura dei resti su S*4

esterna l'esatto gruppo di vettori compressi tenendo conto delle esigenze del processore.

Si supponga in particolare che il program counter (PC) del processore richieda una riga di istruzioni che non è già presente nella rispettiva cache di istruzioni (I$).

Dividendo questo valore per la dimensione della linea di cache (16, negli esempi fatti in precedenza) il de-compressore identifica qual è il gruppo di vettori in cui è presente l'istruzione richiesta. Dividendolo ancora per 2 ottiene anche il gruppo di compressione con il relativo indirizzo della tabella ATT. Se il numero del gruppo di vettori è pari, dovrà leggere soltanto una voce di 26 bit nella tabella ATT. Altrimenti dovrà anche aggiungere l'offset di 6 bit così da ottenere l'indirizzo del gruppo di vettori compresso nella memoria principale.

A questo punto può prelevare il gruppo di vettori compresso e cominciare con la decompressione vera e propria. In primo luogo dovrà leggere i primi 3 bit (bit SS delle figure 8 e 9) per capire quanti shift sono stati fatti. A partire da questo numero leggerà i campi di S bit ciascuno (R1, R2, R3 e R4), ossia l'informazione indicata con SB nelle figure 8 e 9, che rappresentano il resto da aggiungere alla parola decodificata nonché i 5 bit aggiuntivi che rappresentano la norma (campo N delle figure 8 e 9). A questo punto esso può leggere i bit rimanenti rilevati al vettore corrente, ossia la parola codificata vera e propria che può essere compressa con uno dei tre algoritmi che verranno descritti nel seguito, ottenendo quattro piccoli indici.

Per ottenere nuovamente i quattro indici effettivi dovrà shiftare S volte a sinistra (operazione

Lo schema della figura 10 riproduce, in termini affatto generali, uno schema di compressione realizzato secondo l'invenzione.

A partire da un passo iniziale, indicato con 100, in cui la funzione count che abbiamo visto in precedenza viene posta a 0, in una fase o passo di confronto 102 si verifica se il parametro W già introdotto in precedenza sia minore di un valore predeterminato, quale ad esempio 32.

In caso negativo (cosa che avviene normalmente all'inizio dell'applicazione), in una sequenza di passi indicato con 104, 106, 108 e 110 si realizzano le operazioni descritte in precedenza, ossia, previo incremento della funzione count (passo 104), si introduce nello stream il bit meno significativo di ciascuna parola (passo 106), effettuando quindi la divisione per 2 (passo 108), calcolando quindi la norma (passo 110) precedendo in modo iterativo fino a quando la stessa è meno del valore pari a 32 utilizzato nel passo di confronto indicato con 102. Completato il processo iterativo descritto in precedenza, il passo 102 dà esito positivo portando all'emissione del risultato in un passo indicato con 112.

Fatte le suddette considerazioni generali riguardo al processo di compressione, ancora una volta può essere utile, per procedere maggiormente nel dettaglio, affrontare dapprima l'aspetto del processo di decompressione.

Questo sarà fatto facendo riferimento a tre diverse possibili implementazioni del nucleo del decodificatore, che si differenziano una dall'altra per le prestazioni in termini di velocità e di esigenze in termini di onere computazionale/memoria.

In generale, un decompressore operante secondo la presente invenzione deve poter recuperare dalla memoria

In sostanza, i passi indicati con 142 e 144 corrispondono al recupero del gruppo di vettori compressi, alla lettura dell'informazione inerente al numero degli shift effettuati e alla lettura dell'informazione relativa alla norma. A partire da questa informazione, nel passo indicato con 146 vengono quattro campi R1, R2, R3, R4 destinati ad essere aggiunti alla parola decodificata (letta nel passo 148 e decodificata nel passo 150 così da generare gli indici i1, i2, i3 e i4.

Il passo indicato con 152 corrisponde all'effettuazione degli shift in funzione dell'informazione letta nel passo 142 ed il nodo indicato con 154 corrisponde all'operazione di somma che porta alla generazione degli indici i1, i2, i3, i4 prima di pervenire ad un passo di stop 156.

La terza fase del processo di decodifica, rappresentata nella figura 13, corrisponde alla codifica vera e propria.

Dal punto di vista concettuale essa corrisponde all'opposto dell'algoritmo di compressione descritto in precedenza con riferimento alla costruzione della tabella N nel caso in cui i parametri L e K sono, rispettivamente, pari a 4 e 31, così da ottenere una matrice di dimensioni [4, 31].

Partendo dalla parola codificata di ingresso, passo 150 e dal suo valore di norma si procede all'inserimento nell'ultima riga della matrice N nella colonna indicata dalla norma.

Il valore letto nella matrice e il valore di ingresso vengono confrontati in un passo indicato con 152.

Si possono verificare due casi.

Se il valore di ingresso è inferiore a quello letto (esito positivo del passo 152) si procede, in un

equivalente ad una moltiplicazione per 2^s) i quattro numeri e poi aggiungere i resti R1, R2, R3, R4.

La figura 11 rappresenta - con riferimento al livello più elevato - il processo di decompressione.

In sostanza il processo di decompressione prevede lo svolgimento di tre ordini di passi o operazioni.

Il primo è la rivelazione dell'indirizzo del gruppo di vettori di partenza.

Con riferimento alla sequenza già descritta in precedenza, il passo 120 indica la richiesta, da parte del program counter PC di una linea di istruzioni che non è presente nella cache istruzioni.

I passi indicati con 122 e 124 rappresentano appunto la divisione del valore in questione per le dimensioni della riga di cache (16) e l'ulteriore divisione per 2, rispettivamente. L'uscita del passo di divisione 124 corrisponde all'identificazione del gruppo di compressione che consente, in un passo 126, di indirizzare la tabella ATT.

Il passo indicato con 128 è il passo in cui si verifica se il numero identificativo del gruppo di vettori è pari.

Se tale numero è pari (esito positivo del passo 128) nel passo 130 si procede alla lettura della voce di 26 bit nella tabella ATT.

In caso di esito negativo del passo 128 (numero del gruppo di vettori dispari) si procede alla somma dei 6 bit di offset, operazione che viene svolta nel passo indicato con 132.

Il riferimento 134 indica il passo di arresto della procedura.

La figura 12 rappresenta invece la seconda fase del processo di decodifica, corrispondente alla ricostruzione degli indici dei vocabolari di partenza.

un'ulteriore colonna (la sesta) oltre a quelle necessarie così da meglio chiarire il modo di procedere.

L/K	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	1	1	1	1	1
3	1	2	3	4	5	6
4	1	3	6	10	15	21

La prima considerazione che si può fare è data dal fatto che le prime due righe (costituite la prima tutta da zeri e la seconda tutte da uno) non sono necessarie.

Queste righe possono quindi essere rimosse semplificando la matrice, considerando altresì che le ultime due componenti da decodificare possono essere ottenute direttamente dal resto dell'ingresso e dalla sua norma tramite differenza.

Inoltre, le altre L meno due righe possono essere modificate, in modo che possano contenere direttamente le accumulazioni dei numeri ed esplodendole in funzione della norma di ingresso come indicato nella tabella che segue, dove le celle indicate con "BLANK" sono vuote e in realtà non necessarie.

	1	2	3	4	5	6
1	1	BLANK	BLANK	BLANK	BLANK	BLANK
2	2+1=3	2	BLANK	BLANK	BLANK	BLANK
3	5+1=6	3+2=5	3	BLANK	BLANK	BLANK
4	9+1=10	7+2=9	4+3=7	4	BLANK	BLANK
5	14+1=15	12+2=14	9+3=12	5+4=9	5	BLANK
6	20+1=21	18+2=20	15+3=18	11+4=15	6+5=11	6
1'	1	BLANK	BLANK	BLANK	BLANK	BLANK
2'	3+1=4	3	BLANK	BLANK	BLANK	BLANK

passo indicato con 154 a sottrarre al primo valore dal secondo inviando in uscita, su una linea indicata con 156, il valore di indice decodificato. Si procedere quindi, in un passo indicato con 158 a ridurre di uno il valore della norma.

Qualora invece il passo 152 dia esito negativo la differenza fra la norma di partenza e quella effettiva è l'ultimo degli L indici codificati, per cui il processo viene avviato passando, in un passo indicato con 160 dalla riga successiva della colonna là dove il precedente confronto ha dato esito negativo.

Il passo indicato con 162 è semplicemente un passo di confronto legato allo svolgimento in modo ciclico della sequenza dei passi indicati con 152, 154, 158. In particolare, quando si verifica che l'indice corrente C, inizialmente posto al valore L è arrivato al valore 0, si è semplicemente verificato (esito positivo del passo 162) che il procedimento è stato completato, per cui il sistema evolve verso un passo di stop indicato con 164.

La soluzione appena descritta può essere ulteriormente perfezionata in termini di prestazioni sfruttando il parallelismo intrinseco dell'algoritmo di base.

Questo può avvenire secondo le modalità ulteriormente descritte nel seguito.

In particolare, una possibile variante di implementazione del processo di decompressione è basata sulla considerazione che l'anello interno del processo visto in precedenza (passi 152 e 160) può essere notevolmente ridotto in termini di iterazione "esplodendo" la tabella N_table in più righe ed esaminandola con attenzione.

La tabella qui sotto riprodotta costituisce un esempio con L=4 e K pari a 5. E' stata aggiunta

essi, e l'altro altro non è che il rimanente dato di ingresso.

Questi ultimi due indici vengono emessi, invece che dal passo 172, dal passo 174, a partire dal quale il sistema evolve verso un passo finale d'arresto indicato con 176.

La variante di implementazione rappresentata nella figura 15 è per molti versi molto simile a quella rappresentata nella figura 14. Per questo motivo passi identici o equivalenti a quelli già descritti in precedenza con riferimento alla figura 14 sono stati indicati con gli stessi riferimenti nella figura 15.

La differenza principale fra la soluzione illustrata nella figura 15 e la soluzione illustrata nella figura 14 sta nel fatto che nel passo 166' nella figura 15 il sistema sceglie la prima riga non-negativa procedendo da sinistra verso destra così da generare, oltre all'indicazione di posizione sfruttata nel passo 172 per generare gli indici in uscita (eccezion fatta per gli ultimi due) anche il corrispondente valore che viene sfruttato nel passo 168.

Per fare questo, il banco o batteria di comparatori 164 illustrato nello schema della figura 14 è sostituito nello schema della figura 15 da un blocco o batteria di sommatori 164'.

Questa soluzione è preferibile, in funzione della latenza dei sommatori e dei comparatori, e delle relative dimensioni.

La soluzione rappresentata nella figura 16 corrisponde ad ancora un'ulteriore variante in cui tutti i possibili confronti vengono effettuati in parallelo in un colpo solo utilizzando l'informazione proveniente dall'esplorazione della tabella della norma.

3'	9+1=10	6+3=9	6	BLANK	BLANK	BLANK
4'	19+1=20	16+3=19	10+6=16	10	BLANK	BLANK
5'	34+1=35	31+3=34	25+6=31	15+10=25	15	BLANK
6'	55+1=56	52+3=55	46+6=52	36+10=46	21+15=36	21

In questo modo è possibile semplificare notevolmente il decodificatore, confrontando direttamente la parola codificata in ingresso con il numero contenuto nella riga indicata dalla norma di ingresso, così come avviene nello schema rappresentato nella figura 14.

In sostanza, partendo da un passo iniziale 160, in una prima fase, indicata complessivamente con 162, un certo numero di funzioni di comparazione, indicate complessivamente con 164 e pari in numero al valore massimo della norma disponibile (31, nell'esempio principale considerato in precedenza) realizzano il confronto del valore di ingresso con il numero nella tabella N_table. Tutto questo selezionando in un passo di uscita indicato con 166 la prima riga non vera a partire da sinistra verso destra.

Nella seconda fase, indicata complessivamente con 168 si calcola l'indice sulla base della posizione vincente nella tabella ed è possibile continuare in modo iterativo (in funzione di un passo di controllo indicato con 170 con tutti gli altri indici).

I riferimenti 172 e 174 indicano i passi di emissione degli indici.

Si apprezzerà che la situazione è diversa per gli ultimi due indici. In questi casi l'ultima è pari alla differenza fra la parte rimanente della norma e l'ingresso dopo tutte le sottrazioni effettuate su di

O(2)=(TH_LAST(C,pos)+1)*(2*(N+a)*TH_LAST(C,pos))
/2;
O(3)=TH_LAST(C, pos).

E' anche possibile evitare di caricare la seconda tabella denominata (TH_LAST) calcolando direttamente i valori.

In particolare è possibile identificare un numero di gruppo pari alla norma rappresentata (8, nell'esempio) in cui ciascun gruppo è formato dalla componente con lo stesso ultimo valore (nell'esempio, il gruppo 1 è dalla parola codificata 0 alla parola codificata 44, il secondo dalla parola codificata 45 alla 80, e così via).

E' anche possibile notare che all'interno di ciascun gruppo c'é un numero di sottovettori (dove ciascun sottovettore identificato dal numero nella tabella chiamata TH). In particolare, il primo gruppo è formato da 9 sottovettori (pari alla norma +1), il secondo da 8 sottovettori (pari alla norma) e così via fino all'ultimo gruppo.

E' possibile contare tale gruppi dal primo gruppo al K-esimo applicando la seguente equazione

$$\sum_{K+1} subblocks = \sum_{gyp=0}^{K-1} ((N+1)\text{-grp}) = \{N+1 = L, K-1 = k\} = (k+1)\cdot\frac{2L-k}{2} = S$$

Dal numero S è anche possibile identificare tanto il gruppo quanto il sottovettore e quindi ottenere i due valori codificati.

In particolare è possibile determinare a quale gruppo appartiene la parola codificata corrente, contare quante differenze non negative ci sono fra la posizione rivelata indicata con pos (che identifica il sottovettore corrente) e il numero di parole codificate che appartengono a ciascun gruppo che lo precede.

In questo caso l'articolazione su due fasi 162, 168 viene superata.

Per quanto riguarda l'esplorazione della tabella della norma si può considerare l'esempio di una prima tabella TH contenente 16*136 numeri (con una norma massima pari a 15) di cui il più grande è pari a 815v (10 bit) con circa metà della tabella riempita da zeri. Una tabella di questo genere può essere ottenuta supponendo di visualizzare tutte le possibili parole codificate per ciascuna norma possibile in ordine crescente. In questo modo si vede che anche le quattro parole originali che formano quella codificata sono anch'esse ordinate.

In particolare, è possibile identificare un gruppo per ciascun cambiamento del numero di componenti 2 (partendo da 0). Nella tabella TH sopra indicata, che è essenzialmente una tabella di soglia, si scrive ciascun valore di parola codificata che rappresenta l'inizio dei gruppi.

A questo punto risulta però necessaria la seconda tabella, con le stesse dimensioni di quella descritta in precedenza, che rappresenta il valore reale della parola denominata 3.

A questo punto, riferendosi alla figura 16, è possibile determinare a partire da una certa parola codificata, il suo valore originale per parole del tipo (O(0:3)).

Questo può essere fatto nel modo seguente.

Indicando con C la differenza fra la massima norma disponibile e la norma reale N della parola codificata e "pos" l'indice nella prima tabella del gruppo dove è collocata la parola codificata in ingresso è possibile definire le seguenti grandezze.

O(0)0TH(C,pos+1)-input-1;

O(1)=input-TH(C,pos);

di cache, fra il primo ed il secondo livello, così da mascherare direttamente la sua latenza.

In alternativa, così come rappresentato nella figura 18, un decodificatore secondo l'invenzione, indicato con D2 è collocato direttamente al terminale di uscita della memoria esterna MP.

Si apprezzerà che in ogni caso la soluzione secondo l'invenzione consente di superare le limitazioni più rilevanti oggi esistenti in relazione ai sistemi di tipo embedded ossia costi, assorbimento/dissipazione di potenza e ingombro.

La soluzione secondo l'invenzione consente di ridurre l'occupazione di un tale sistema, riducendone le esigenze in termini di memoria. La soluzione secondo l'invenzione consegue altresì una riduzione dell'assorbimento/dissipazione di potenza sia per il fatto di ridurre le dimensioni e dunque l'ingombro della memoria di programma, sia perché riduce l'attività di commutazione (SA) del bus di memoria.

Naturalmente, fermo restando il principio dell'invenzione, i particolari di realizzazione e le forme di attuazione potranno essere ampiamente variati rispetto a quanto descritto ed illustrato, senza per questo uscire dall'ambito della presente invenzione, così come definita dalle rivendicazioni annesse.

Questo può essere fatto con la formula che permette di calcolare S.

Il numero di gruppo identica l'ultima parola non compressa. Per ottenere la parola numero due si deve applicare la formula

$$O(2) = grp \cdot \frac{2 \cdot L - grp + 1}{2}$$

Nello schema della figura 16, i riferimenti 164'' indicano rispettivi comparatori che effettuano il confronto sul valore di TH così da selezionare, nel passo indicato con 166 la prima riga da sinistra verso destra.

I riferimenti 180 a 188 identificano i passi in cui vengono implementate le relazioni citate in precedenza.

Infine, le figure 17 e 18 illustrano due possibili architetture di sistema processore suscettibile di impiegare la soluzione secondo l'invenzione.

In entrambe le figure è rappresentato un processore, indicato come CPU con associate rispettive cache rispettivamente di dati D$ e istruzioni I$. La linea su cui il processore eroga verso la cache istruzioni I$ il program counter PC è indicata con il simbolo omologo.

Si suppone poi che il processore CPU si interfacci tramite una normale interfaccia CMC con un bus B che gli consente di colloquiare con una memoria principale MEM con una o più periferiche P1, P2, ... Pn con una memoria di programma PC.

Nell'architettura rappresentata nella figura 17, un decodificatore secondo l'invenzione, indicato con D1, tipicamente configurato come un piccolo ASIC, può essere collocato fra il bus di sistema B (e la cache istruzioni I$). Questo può avvenire tramite collocazione nel cosiddetto memory control unit o, se è presente la gerarchia di cache con almeno due livelli

vettori (GV) con dimensioni pari alla cache di riga del processore, per cui ciascun vettore (V) contiene un numero di indici (nI) pari al rapporto fra le dimensioni di detta riga di cache ed il numero (B) di vettori contenuti in ciascun gruppo.

3. Procedimento secondo la rivendicazione 2, caratterizzato dal fatto che tanto il numero (B) di vettori compresi in ciascun gruppo quanto il numero di indici (nI) compresi in ciascun vettore sono entrambi potenza di due.

4. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, caratterizzato dal fatto che comprende l'operazione di suddividere dette parole binarie in detto formato non-codificato in un numero dato (d) di sotto-parole (n_1, n_2, ..., n_m), dette sotto-parole binarie essendo convertite in detto formato codificato ricorrendo a rispettivi vocabolari di codifica.

5. Procedimento secondo la rivendicazione 4, caratterizzato dal fatto che comprende l'operazione di suddividere dette parole binarie in due sotto-parole binarie, corrispondenti rispettivamente alla parte alta ed alla parte bassa della parola binaria nel formato non-codificato.

6. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, caratterizzato dal fatto che comprende l'operazione di costituire detti vocabolari di codifica riservando a parole binarie di bassa ricorrenza rispettive sequenze di bit codificate comprendenti un numero di bit suscettibile di risultare non inferiore rispetto al numero di bit compresi nella corrispondente parola binario in formato non-codificato.

7. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, caratterizzato dal fatto che

RIVENDICAZIONI

1. Procedimento per convertire parole binarie fra un formato non-codificato (OP) e un formato codificato compresso (V), in cui, in detto formato codificato compresso, delle parole binarie sono, almeno in parte, rappresentate da sequenze di bit codificate più corte della rispettiva parola binaria nel formato non-codificato, dette sequenze di bit codificate essendo selezionate in funzione della ricorrenza statistica delle rispettive parole nel formato non-codificato ed in cui alle parole binarie con più elevata ricorrenza sono associate sequenze di bit codificate comprendenti numeri di bit corrispondentemente più ridotti, la corrispondenza fra parole binarie in formato non-codificato e le sequenze di bit codificate ad esse associate essendo stabilità tramite indici di almeno un vocabolario di codifica, caratterizzato dal fatto che comprende le operazioni di:

- ordinare detti indici in una sequenza ordinata,
- organizzare detta sequenza di indici in gruppi di vettori (GV),
- suddividere ciascun gruppo di vettori in un numero dato di vettori (V),

ed almeno una fra le operazioni di:

- codificare detti vettori (V) in modo indipendente l'uno dall'altro,
- calcolare e memorizzare (ATT), per ciascun gruppo di vettori (GV), l'indirizzo di partenza del blocco compresso, oppure le differenze rispetto all'ultimo indirizzo completo riportatovi.

2. Procedimento secondo la rivendicazione 1, applicato a parole binarie costituenti le istruzioni di un processore avente una cache di riga di dimensione data (CACHE_LINE), caratterizzato dal fatto che comprende l'operazione di formare detti gruppi di

binaria nel formato non-codificato, tale operazione di shift essendo attuata sino a quando la corrispondente norma raggiunge un valore predeterminato.

13. Procedimento secondo la rivendicazione 12, caratterizzato dal fatto che detta sequenza di bit codificata è organizzata come insieme di campi comprendenti:

- un campo identificativo del numero di shift realizzati (SS),
- l'identificazione dei bit sottoposti a shift (SB),
- il valore di norma risultante (N), e
- detta sequenza di bit codificata più corta (EW).

14. Procedimento secondo la rivendicazione 13, caratterizzato dal fatto che comprende l'operazione di collocare detto campo identificativo dei bit sottoposti a shift (SB) al fondo della sequenza di bit codificata più corta corrispondente.

15. Procedimento secondo una qualsiasi delle precedenti rivendicazioni 9 a 14, caratterizzato dal fatto che comprende le operazioni di:

- definire un numero (L) di indici successivi suscettibili di essere codificati insieme come una singola parola codificata,
- costituire una tabella di confronto o look-up con dimensioni LxK, dove L è detto valore pari al numero delle parole destinate ad essere concatenate insieme e K rappresenta la massima norma possibile per le parole codificate, determinando quindi la parola codificata a partire da L successivi indici secondo la relazione

$$encoded = \sum_{i=1}^{L} \sum_{j=1}^{x} Ntable[i,j]$$

dove *encoded* rappresenta la parola codificata, le sommatorie si estendono rispettivamente per i che va da

comprende l'operazione di memorizzare detti gruppi di vettori (GV) memorizzando, per ciascuno di detti gruppi di vettori (GV), un rispettivo indirizzo di partenza.

8. Procedimento secondo la rivendicazione 7, caratterizzato dal fatto che comprende l'operazione di provvedere una tabella di traduzione degli indirizzi (ATT) per salvare i suddetti indirizzi di partenza di detti gruppi di vettori.

9. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, caratterizzato dal fatto che comprende l'operazione di determinare, per ciascuno di detti vettori (V), una corrispondente norma data dalla somma degli indici contenuti nel vettore stesso.

10. Procedimento secondo la rivendicazione 9, caratterizzato dal fatto che comprende l'operazione di sottoporre detta norma ad una codifica di lunghezza variabile (VLC).

11. Procedimento secondo la rivendicazione 10, caratterizzato dal fatto che detta norma viene codificata con una stringa di bit comprendente un campo di etichetta o tag ed un campo di bit rappresentativi del valore reale della norma, i valori di norma con più elevata ricorrenza essendo rappresentati con campi di etichetta e campi di rappresentazione del valore reale più corti rispetto ai campi di etichetta ed ai campi di rappresentazione del valore reale riservate ai valori di norma con più ridotta ricorrenza statistica.

12. Procedimento secondo la rivendicazione 9, caratterizzato dal fatto che per convertire dette parole binarie da detto formato non-codificato in detto formato codificato si procede attribuendo i bit meno significativi di ciascuna parola binaria non-codificata alla corrispondente sequenza di bit codificata procedendo ad un insieme di operazioni di divisione per due corrispondenti a shift di bit compresi nella parola

18. Procedimento secondo la rivendicazione rivendicazione 17, caratterizzato dal fatto che comprende le operazioni di:

- leggere detto gruppo codificato di bit ed il relativo valore di norma,
- collocarlo nell'ultima riga di detta matrice nella colonna identificata dal valore di norma,
- confrontare il valore letto nella matrice e il valore codificato di ingresso,
- se il valore di ingresso è inferiore rispetto al valore letto, sottrarre il valore di ingresso dal valore letto e ridurre di uno il valore della norma fino al soddisfacimento del confronto, e
- se il valore di ingresso è maggiore rispetto al valore letto la differenza fra la norma di partenza e quella effettiva è l'ultimo degli indici codificati, ed il processo viene riavviato dalla riga successiva in corrispondenza della colonna dove il precedente confronto non è stato soddisfatto.

19. Procedimento secondo la rivendicazione 17 o la rivendicazione 18, caratterizzato dal fatto che comprende le operazioni di:

- confrontare direttamente la parola codificata di ingresso con il numero contenuto nella riga indicata dalla norma di ingresso,
- calcolare l'indice risultante da utilizzare per la decodifica, sulla base della posizione vincente nell'ambito della tabella così identificata.

20. Procedimento secondo la rivendicazione 19, caratterizzato dal fatto che, per ottenere gli indici da utilizzare la codifica, si procede ricavando l'ultimo indice come pari alla differenza fra la parte rimanente della norma ed il valore dell'ingresso dopo tutte le sottrazioni effettuate sugli stessi, ricavando

1 a L e per j che va da 1 a x_i e Ntable[i,j] rappresenta l'insieme dei punti a coordinata intera che identificano una piramide ad i dimensioni con norma j, detta norma esprimendo la somma dei valori assoluti delle coordinate dei punti che la compongono.

16. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, caratterizzato dal fatto che dette sequenze di bit codificate mappano al loro interno un indirizzo identificativo della corrispondente parola binaria in formato non-codificato, per cui ciascuna di dette sequenze di bit codificate è suscettibile di essere convertita nella corrispondente parola binaria in formato non-codificato in modo indipendente da altre sequenze di bit codificate.

17. Procedimento secondo una qualsiasi della rivendicazioni 12 a 16, caratterizzato dal fatto che, per convertire una determinata parola binaria da detto formato codificato in detto formato non-codificato, comprende le operazioni di:

- identificare il corrispondente gruppo di vettori (GV) nel formato codificato,
- leggere, a partire da detto gruppo di vettori, detto campo identificativo di quanti shift sono stati realizzati (SS),
- leggere l'informazione relativa ai bit sottoposti a shift (SB),
- leggere l'informazione relativa a detta norma (N),
- leggere l'informazione relativa a detto blocco di informazione codificato (EW), e
- costruire detta parola binaria in detto formato non-codificato a partire dall'informazione letta (SS, SB, N, EW).

1 a 21, detto modulo convertitore (D1, D2) è collocato in almeno una posizione intermedia fra:

- una di dette memorie cache (IS$) e detto bus (B), e
- detto bus (B) ed almeno una di dette memorie (MP).

23. Sistema secondo la rivendicazione 22, caratterizzato dal fatto che detto almeno un modulo convertitore (D1) risulta interposto fra la cache istruzioni (I$) e detta interfaccia di bus (B).

24. Sistema secondo la rivendicazione 22 o la rivendicazione 23, caratterizzato dal fatto che detto almeno un modulo convertitore (D2) è interposto fra detto bus (B) ed una memoria di programma (MP) associata a detto processore.

25. Sistema secondo la rivendicazione 22, caratterizzato dal fatto che detta almeno una memoria cache (I$) è una cache organizzata su due livelli e dal fatto che detto modulo convertitore (D1) è interposto fra detti due livelli di livelli di cache.

26. Prodotto informatico direttamente caricabile nella memoria di un processore digitale e comprendente porzioni di codice software suscettibili di attuare il procedimento secondo una qualsiasi delle rivendicazioni 1 a 18 quando è eseguito su detto processore digitale.

invece il penultimo indice come valore di ingresso residuo.

21. Procedimento secondo la rivendicazione 17, caratterizzato dal fatto che le operazioni di:

- formare una tabella (TH) che raccoglie tutte le parole codificate possibili per detto valore di norma in ordine crescente, per cui anche le parole originali che formano la parola codificata risultano ordinate fra loro,

- determinare, a partire da una certa parola codificata, la corrispondente parola binaria in formato non-codificato sulla base della differenza (C) fra la massima norma disponibile e la norma reale (N) della parola codificata, definendo un indice ("pos") nella prima tabella del gruppo dove la parola di ingresso è collocata,

- determinare quale gruppo appartiene la parola codificata corrente conteggiando quante differenze non-negative esistono per la posizione identificata da detto indice, che identifica il sotto-vettore corrente, ed il numero di parole codificate che appartengono a ciascun gruppo precedente.

22. Sistema processore comprendente un'unità elaborativa (CPU), rispettive memorie cache (I$, D$), detta unità elaborativa (CPU) essendo configurate per poter dialogare tramite un bus (B) con almeno una memoria (MEM, MP) ed almeno un'unità periferica (P1, P2, ..., Pn), a detta unità elaborativa (CPU) essendo associata una corrispondente interfaccia di bus (CMC), il sistema essendo caratterizzato dal fatto che comprende almeno un modulo convertitore (D1, D2) suscettibile di convertire parole binarie fra un formato non-codificato ed un formato codificato con il procedimento secondo una qualsiasi delle rivendicazioni

RIASSUNTO

Parole binarie sono convertite fra un formato non-codificato (OP) e un formato codificato compresso (V), in cui le parole binarie sono, almeno in parte, rappresentate da sequenze di bit codificate più corte della rispettiva parola binaria nel formato non codificato. Le sequenze di bit codificate più corte sono selezionate in funzione della ricorrenza statistica delle rispettive parole nel formato non-codificato ed alle parole binarie con più elevata ricorrenza sono associate sequenze di bit codificate comprendenti numeri di bit corrispondentemente più ridotte. La corrispondenza fra parole binarie in formato non-codificato e le sequenze di bit codificate ad esse associate è stabilità tramite indici di un vocabolario di codifica. Il procedimento di conversione comprende le operazioni di:

- ordinare gli indici in una sequenza ordinata,
- organizzare la sequenza di indici in gruppi di vettori (GV),
- suddividere ciascun gruppo di vettori in un numero dato di vettori (V), e
- codificare i vettori (V) in modo indipendente l'uno dall'altro.

Alternativamente, per ciascun gruppo di vettori, alla fine del processo di codifica si calcola e si salva in una tabella, chiamata tabella di traduzione degli indirizzi (ATT), l'indirizzo di partenza su 32 bit del blocco compresso, oppure le differenze, espresse in byte, rispetto all'ultimo indirizzo completo riportatovi.

(Figura 2)

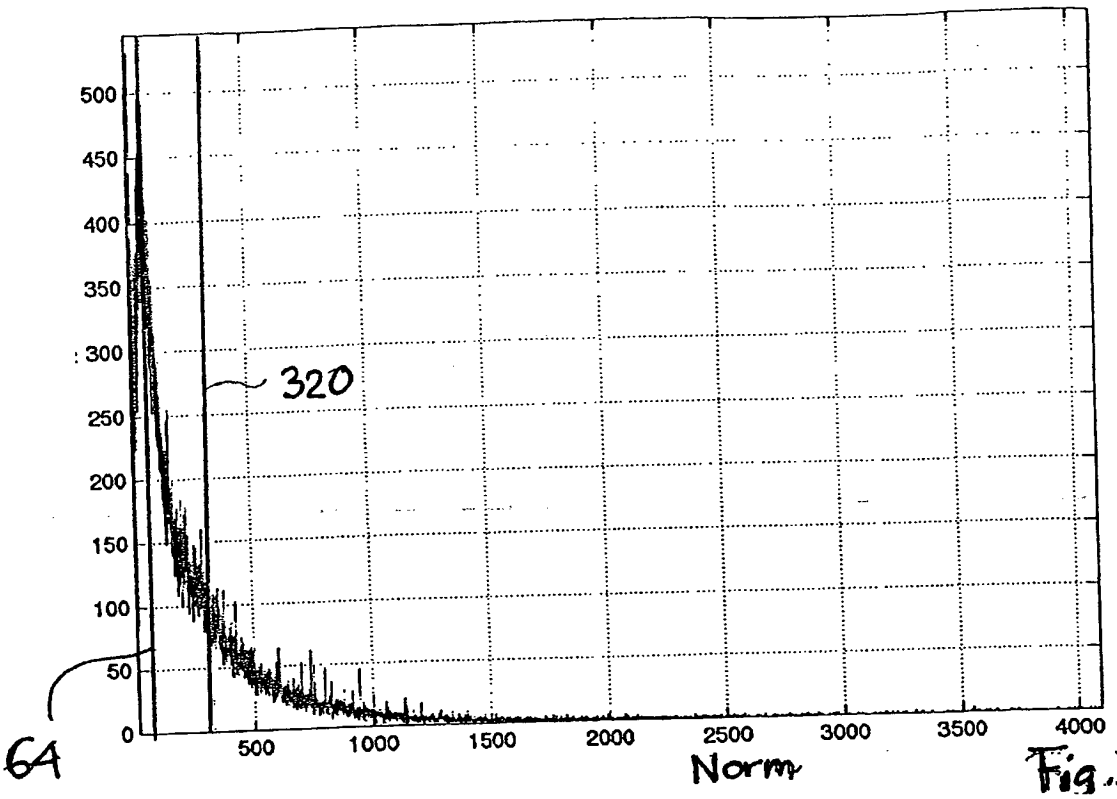

500
450
400
350
300
250
200
150
100
50
0
500
1000
1500
2000
2500
3000
3500
4000
320
64
Norm

Fig. 3

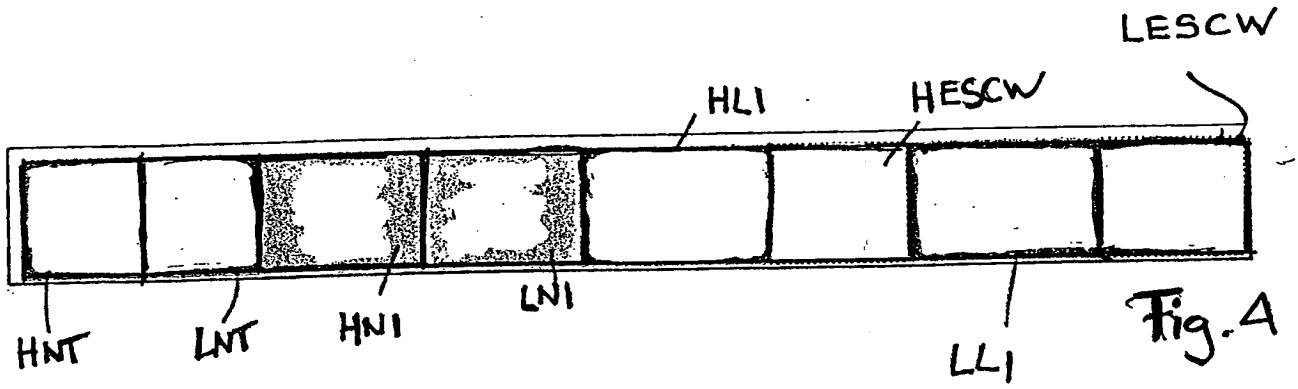

LESCW
HLI
HESCW
HNT
LNT
HNI
LNI
LLI

Fig. 4

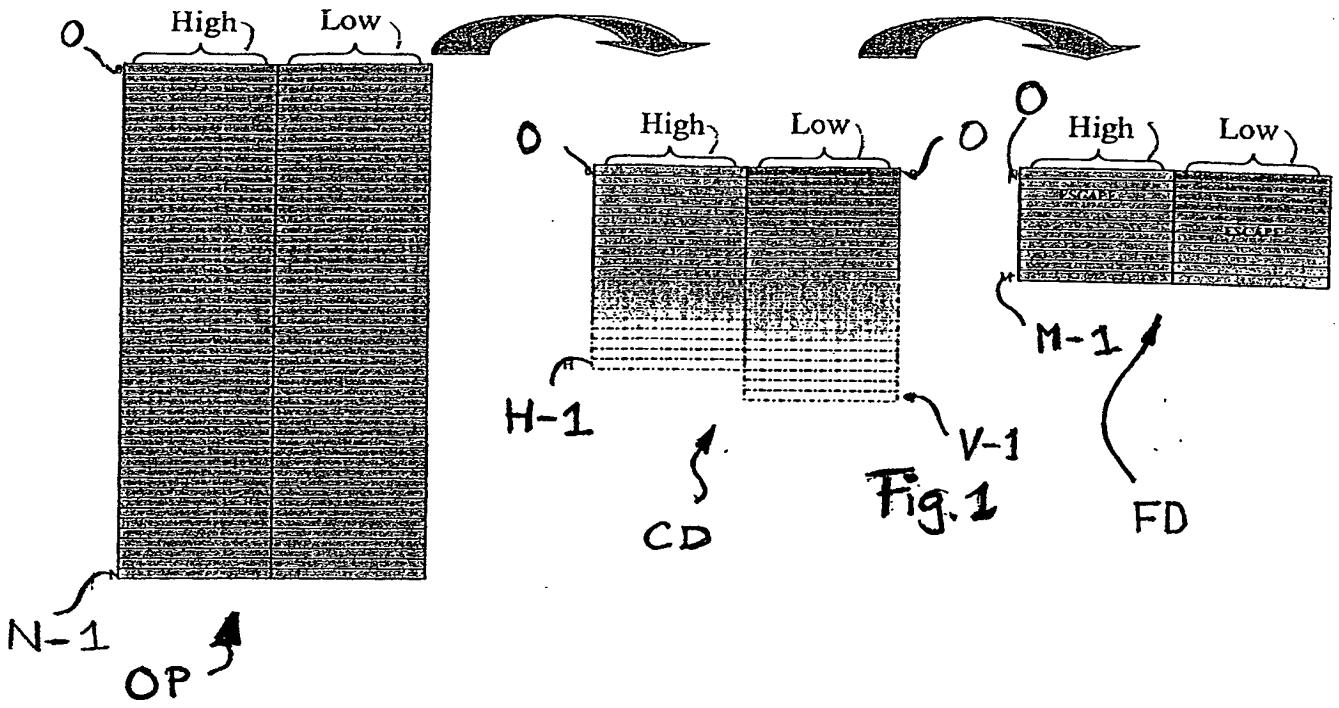

0
High
Low
N-1
OP
0
High
Low
H-1
V-1
CD
0
High
Low
M-1
FD

Fig.1

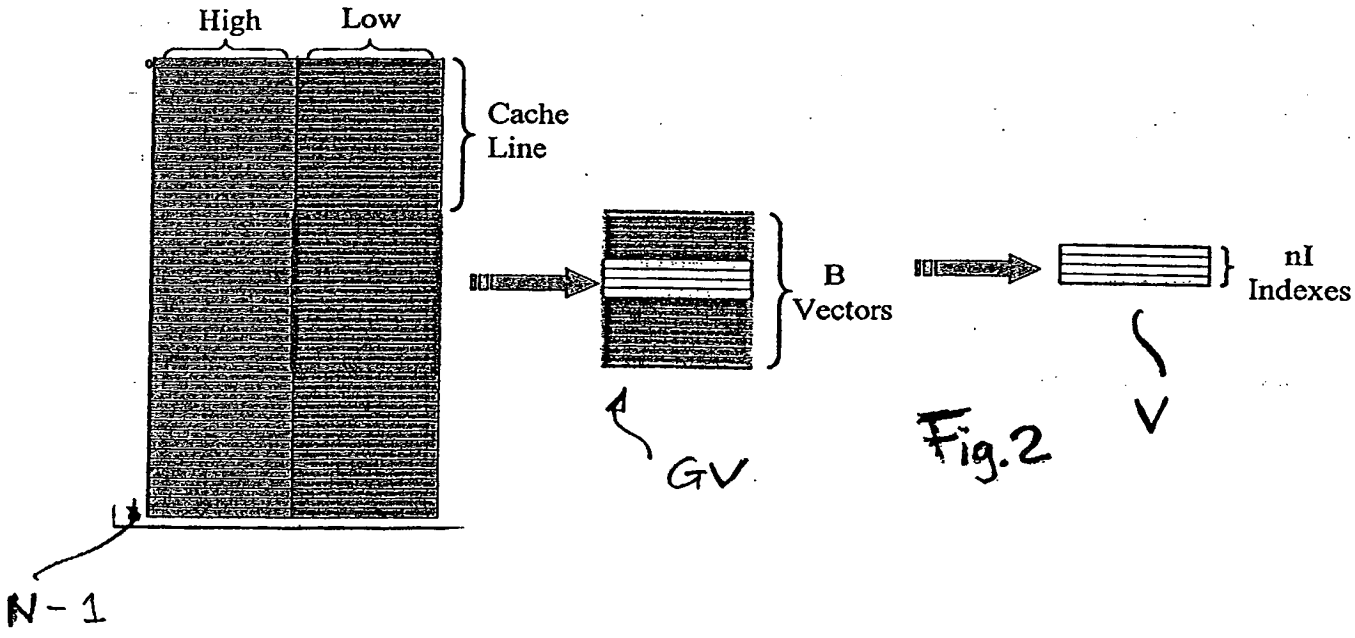

High
Low
Cache Line
N-1
B Vectors
GV
nI Indexes
V

Fig.2

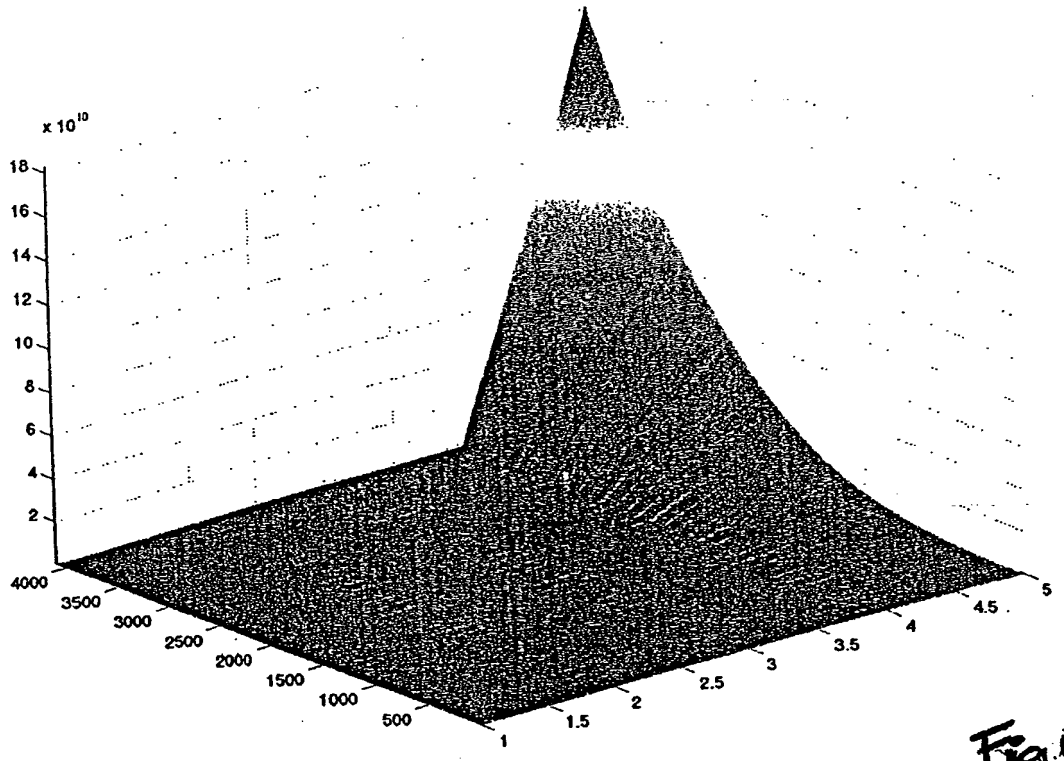

x 10^10
18
16
14
12
10
8
6
4
2
4000
3500
3000
2500
2000
1500
1000
500
1
1.5
2
2.5
3
3.5
4
4.5
5

Fig. 6

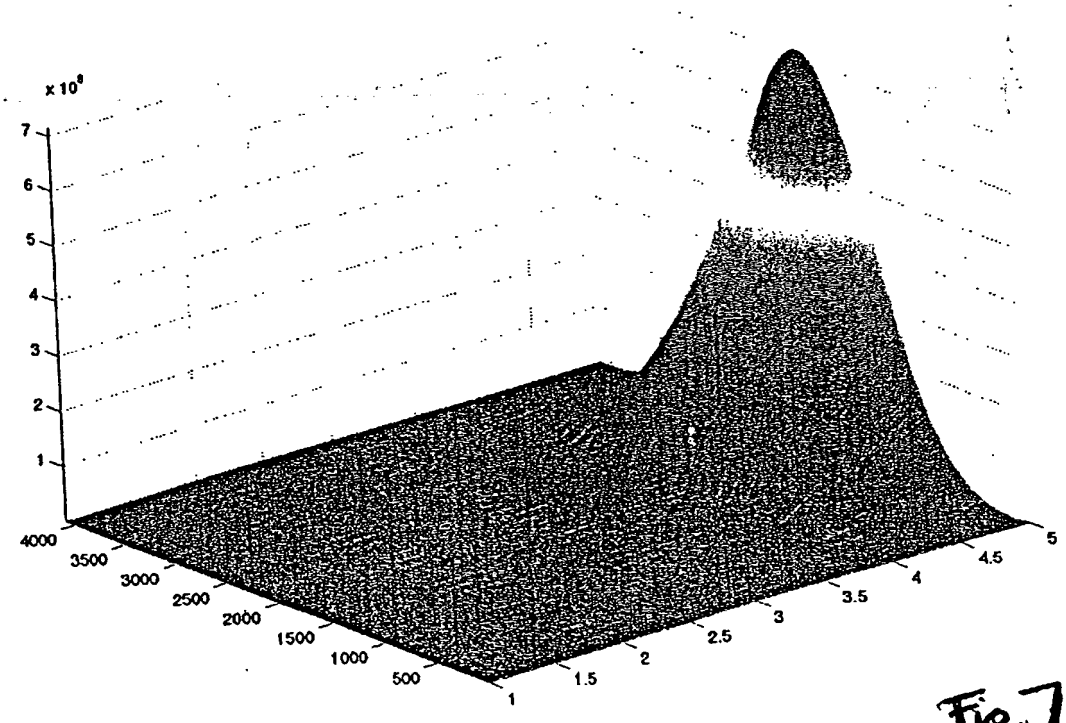

x 10^9
7
6
5
4
3
2
1
4000
3500
3000
2500
2000
1500
1000
500
1
1.5
2
2.5
3
3.5
4
4.5
5

Fig. 7

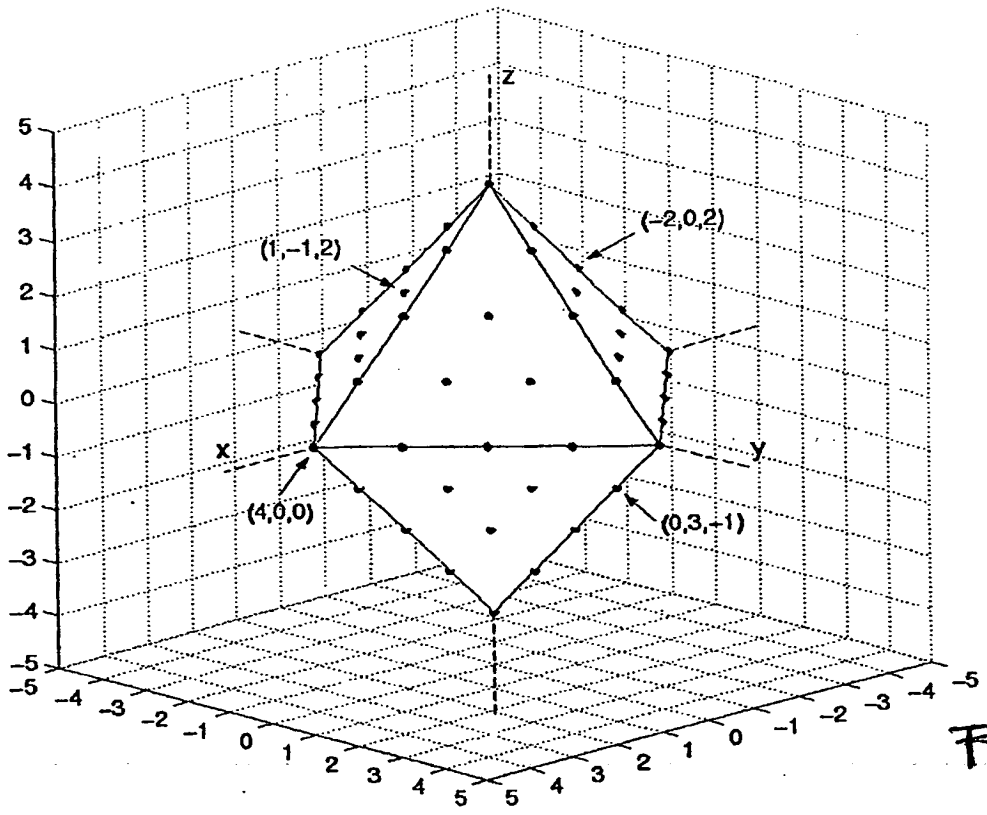


Fig.5

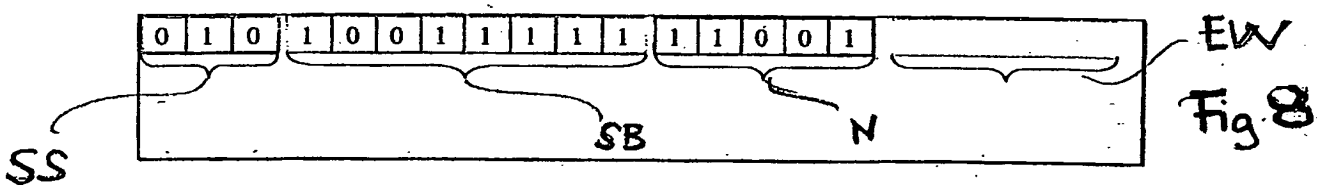


Fig.8

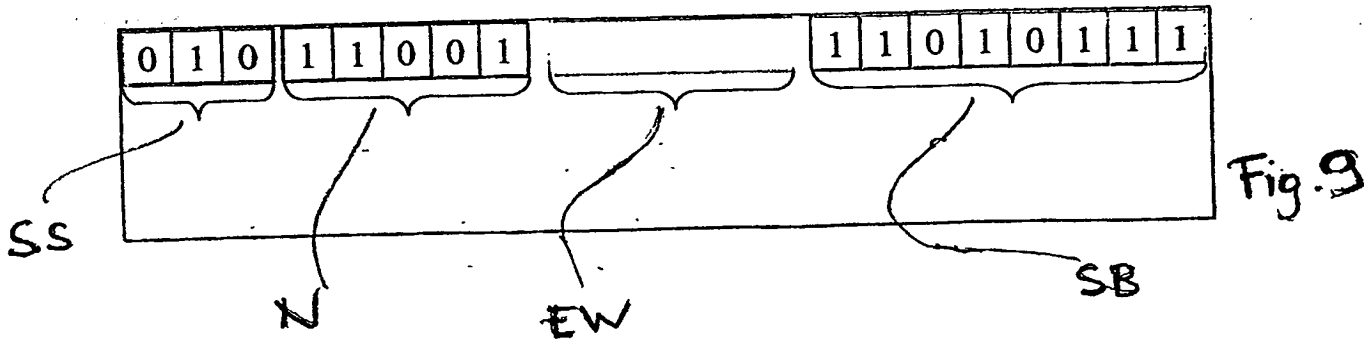


Fig.9

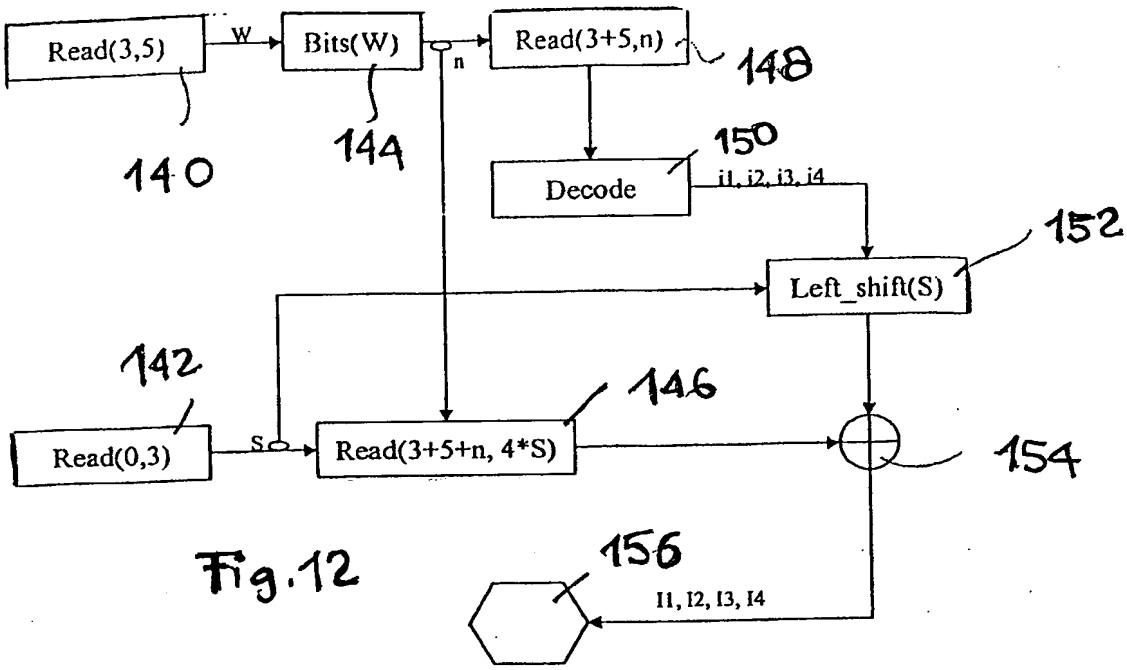

Read(3,5)
W
Bits(W)
n
Read(3+5,n)
148
140
144
150
Decode
i1, i2, i3, i4
152
Left_shift(S)
142
146
Read(0,3)
S
Read(3+5+n, 4*S)
154
156
I1, I2, I3, I4

Fig.12

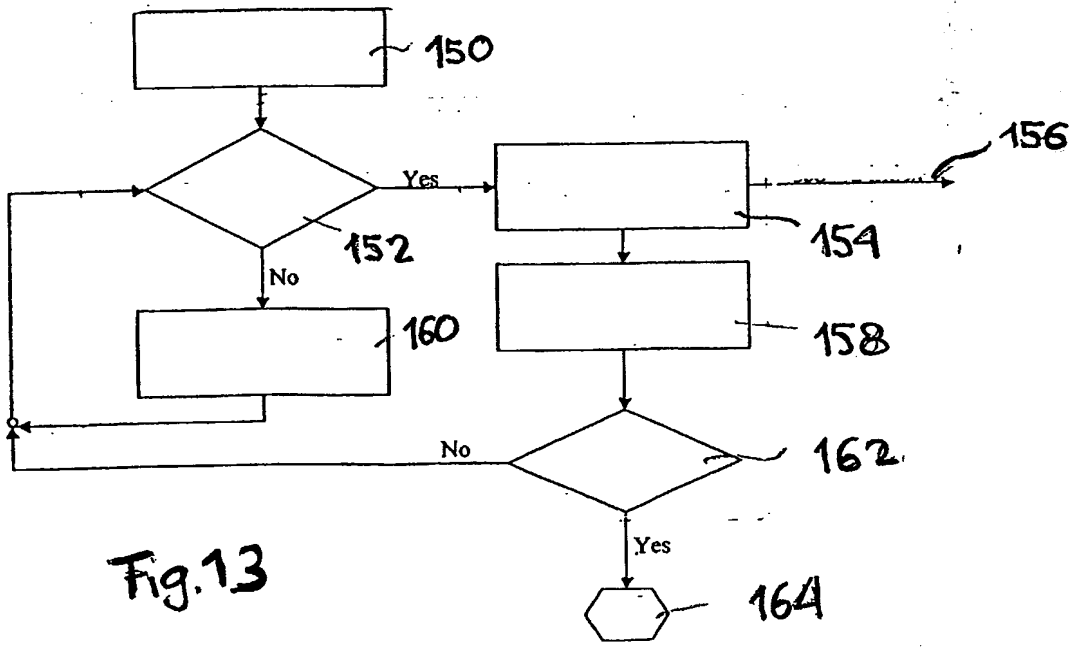

150
156
Yes
152
154
No
160
158
No
162
Yes
164

Fig.13

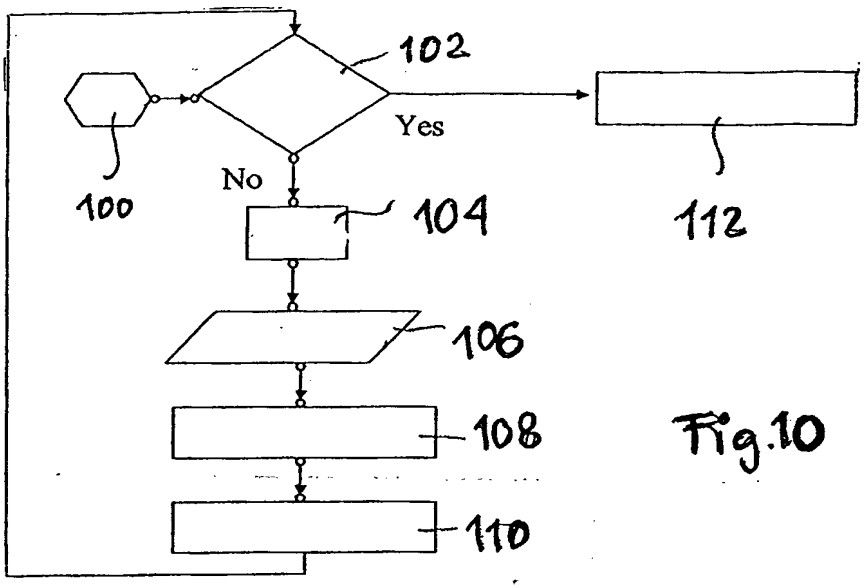

102
Yes
No
100
104
112
106
108
110

Fig.10

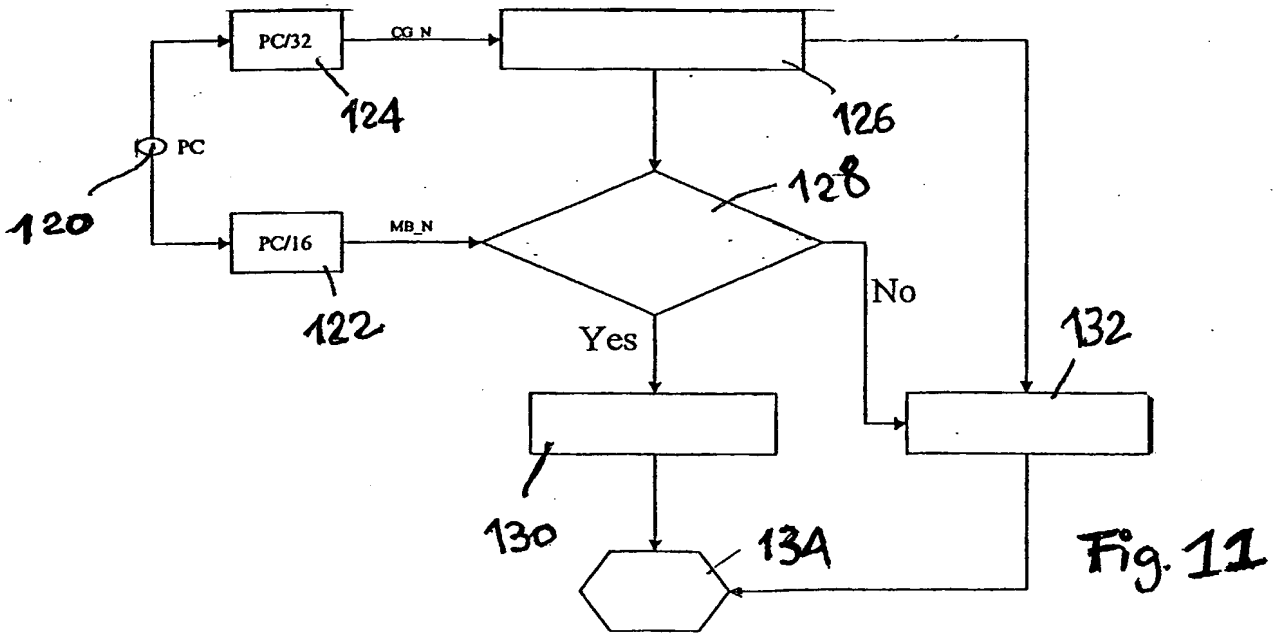

PC/32
CG_N
124
126
PC
120
128
PC/16
MB_N
122
No
Yes
132
130
134

Fig. 11

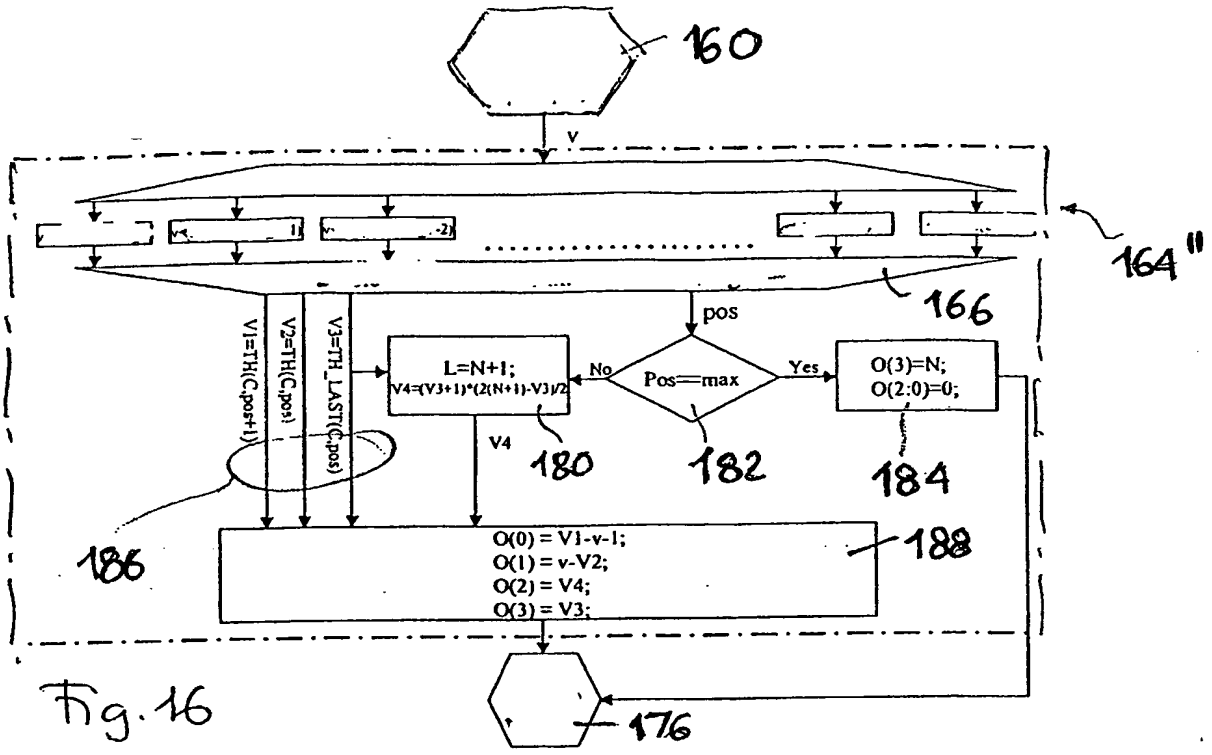

160
V
164"
166
pos
V1=TH(C,pos+1)
V2=TH(C,pos)
V3=TH_LAST(C,pos)
L=N+1;
V4=(V3+1)*(2(N+1)-V3)/2
No
Pos=max
Yes
O(3)=N;
O(2:0)=0;
V4
180
182
184
186
O(0) = V1-v-1;
O(1) = v-V2;
O(2) = V4;
O(3) = V3;
188
176

Fig. 16

Fig.14

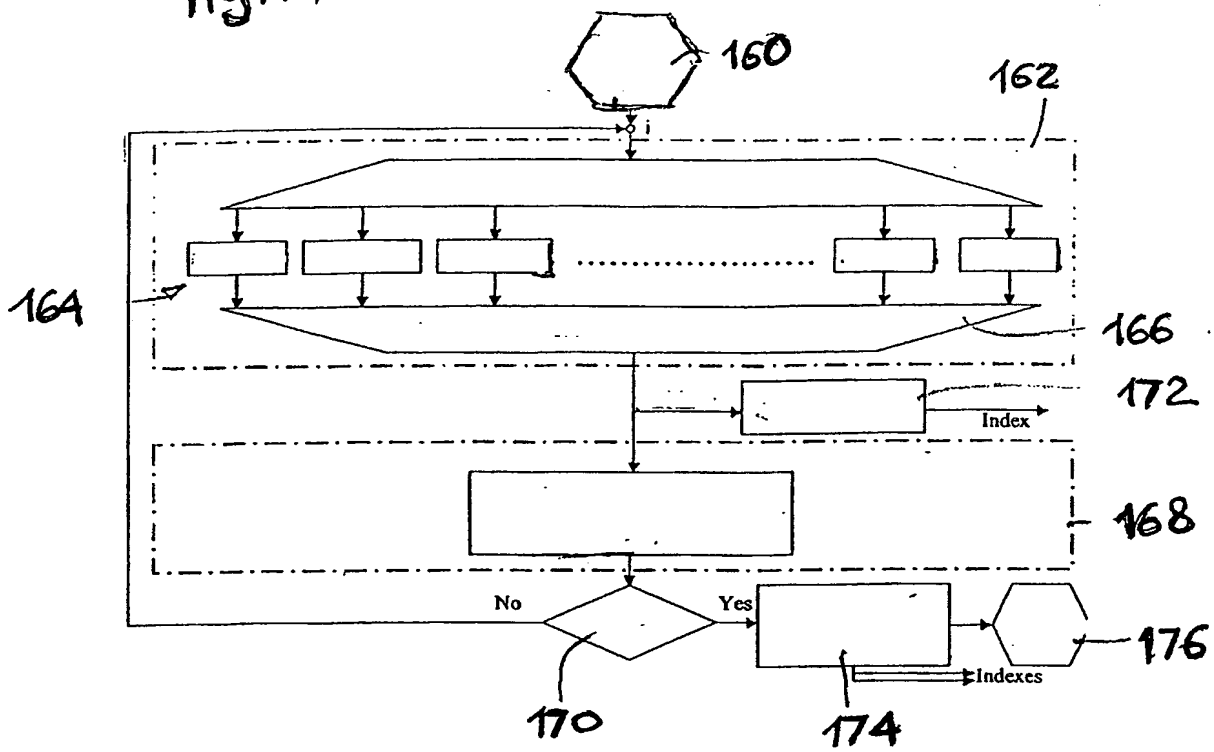
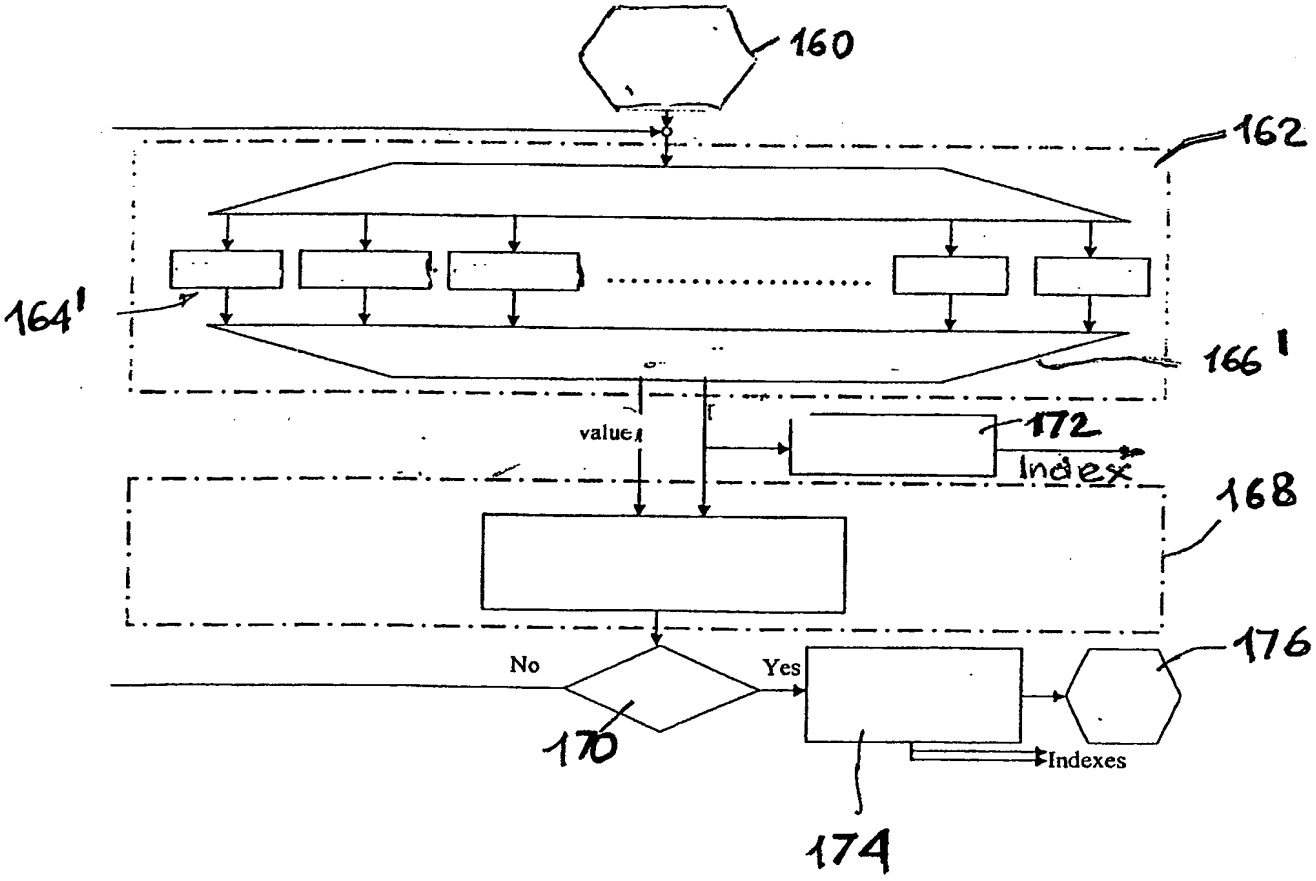

160
162
164
166
172
Index
168
No
Yes
176
Indexes
170
174

Fig.15


160
162
164'
166'
value
172
Index
168
No
Yes
176
Indexes
170
174

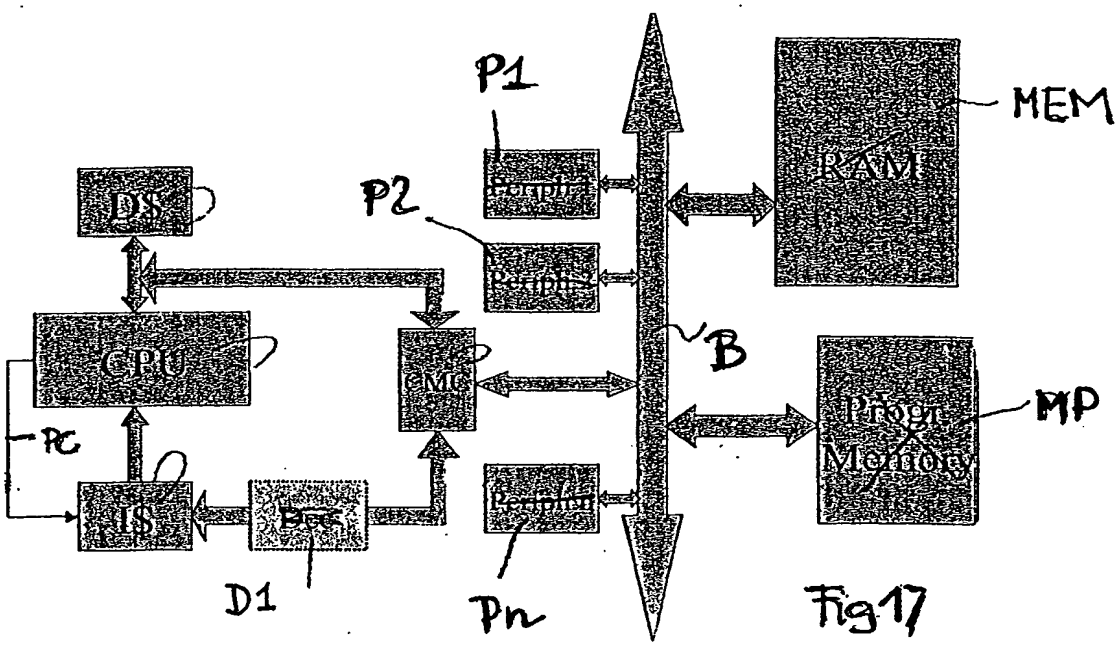

P1
MEM
RAM
P2
DS
Periph 1
Periph 2
CPU
CMC
B
MP
Progr. Memory
PC
I$
Bcs
Periphn
D1
Pn

Fig17

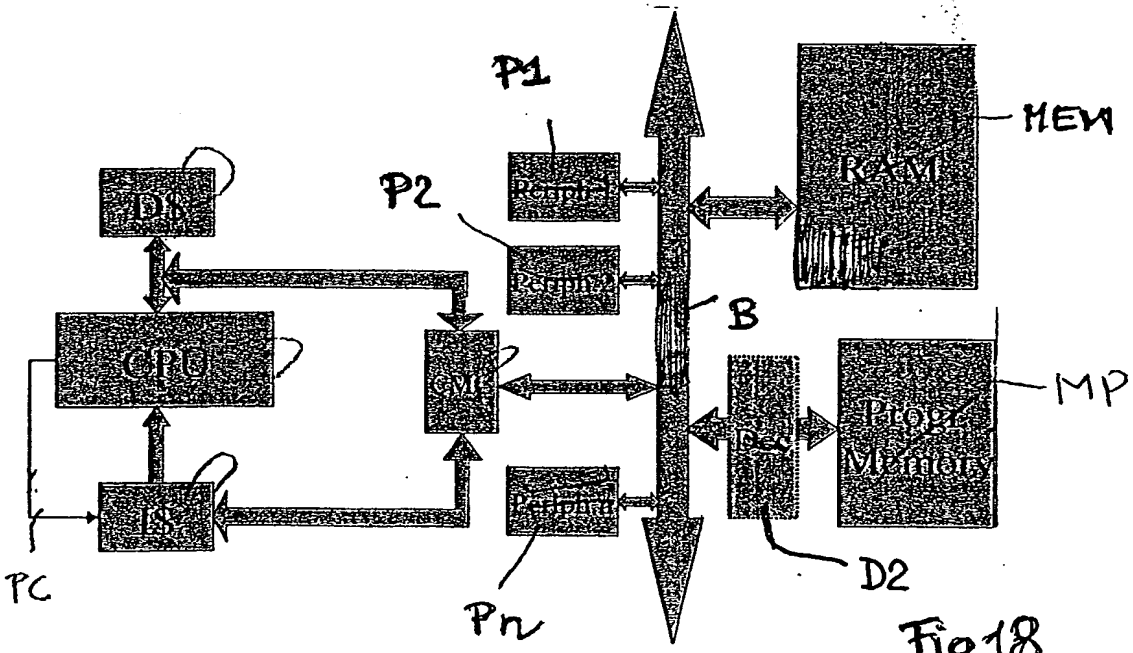

P1
MEM
RAM
P2
D$
Periph 1
Periph 2
B
CPU
CMC
MP
Progr. Memory
Bcs
I$
Periphn
PC
D2
Pn

Fig18